MPLAB Harmony Help

MPLAB Harmony Integrated Software Framework

Volume V: MPLAB Harmony Framework Reference

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

Description

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

Framework Overview

This section provides an overview of the MPLAB Harmony Framework.

Introduction

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

Description

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

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

Bluetooth Stack Library Help

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

Description

The PIC32 Bluetooth Stack Library consists of three distinct libraries:

- *PIC32 Bluetooth Basic Stack Library* – this library provides "Core" Bluetooth functionality, which was previously referred to as "SPP-only"
- *PIC32 Bluetooth Audio Stack Library* – this library, which must be purchased, adds audio features to the SPP-only library
- *PIC32 Bluetooth Audio/HFP Stack Library* – this library, which must be purchased, add hands free and phone features to the audio library

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

The stack works with the following HCI radio modules:

- CSR881x/BTM805
- TI2564b/PAN13xx
- RDA
- IS1662/MFR4662

Introduction

Describes the PIC32 Bluetooth Stack Library.

Description

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

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

PIC32 Bluetooth Stack Library

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

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

PIC32 Bluetooth Audio Stack Library

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

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

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

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

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

Demonstrations

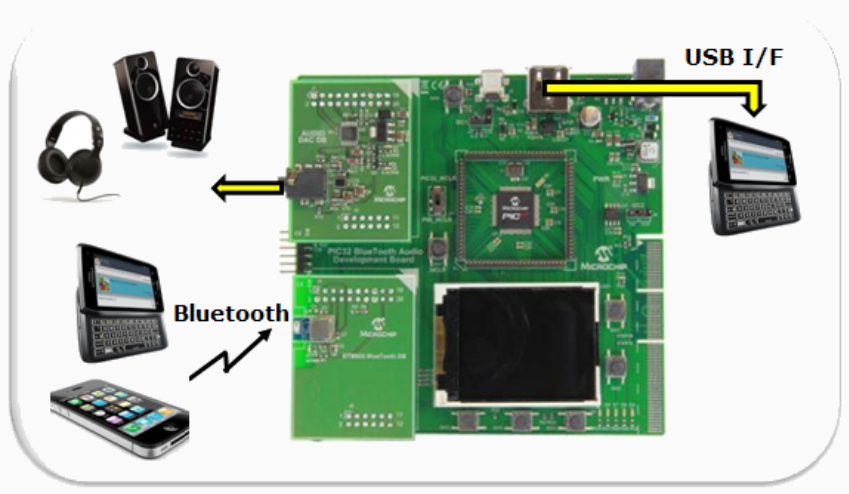
Refer to PIC32 Bluetooth Stack Library Demonstrations for information on the demonstrations that are available for the PIC32 Bluetooth Stack Library. In addition, other specific demonstrations to illustrate the various connection options, data and audio communications and control interfaces are available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

Bluetooth Stack Licensing

This section describes how to license the stack.

Description

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



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

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

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

Using the Library

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

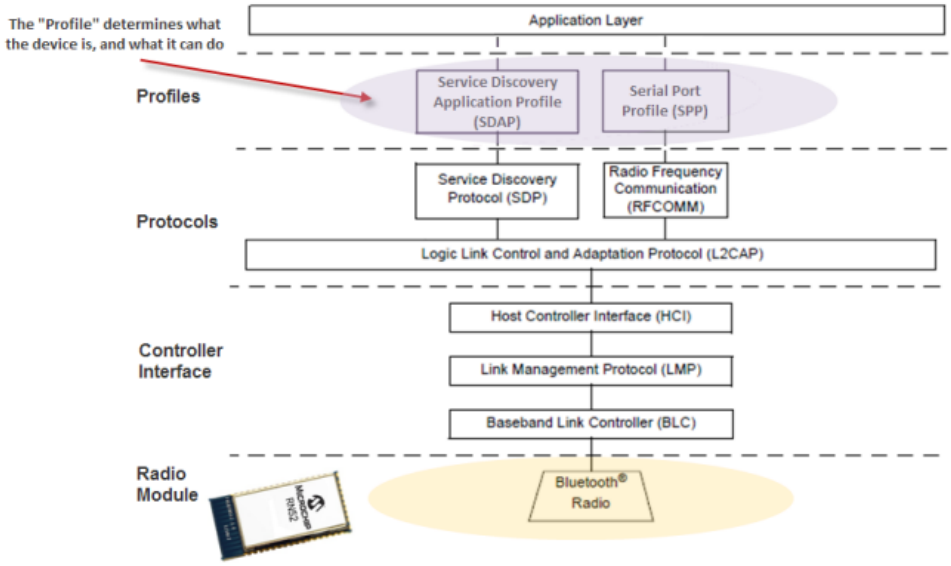
Description

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

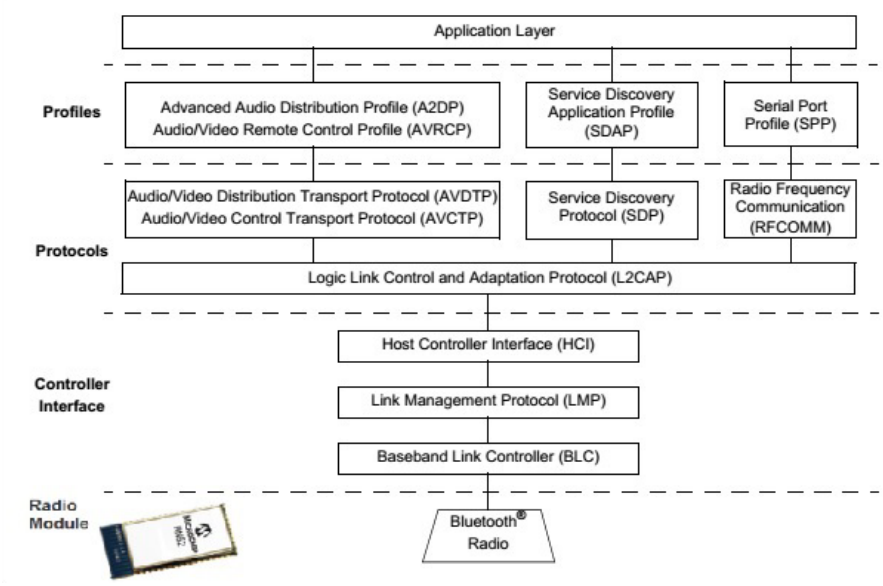
- `<install_dir>/framework/bluetooth/cdbt` for the PIC32 Bluetooth Basic Stack
- `<install_dir>/framework/bluetooth/premium/audio/cdbt` for the PIC32 Bluetooth Audio Stack

The following two figures illustrate the Bluetooth Stacks.

PIC32 Bluetooth Basic Stack



PIC32 Bluetooth Audio Stack



Library Overview

Provides information on the APIs provided in the library.

Description

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

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

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

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

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

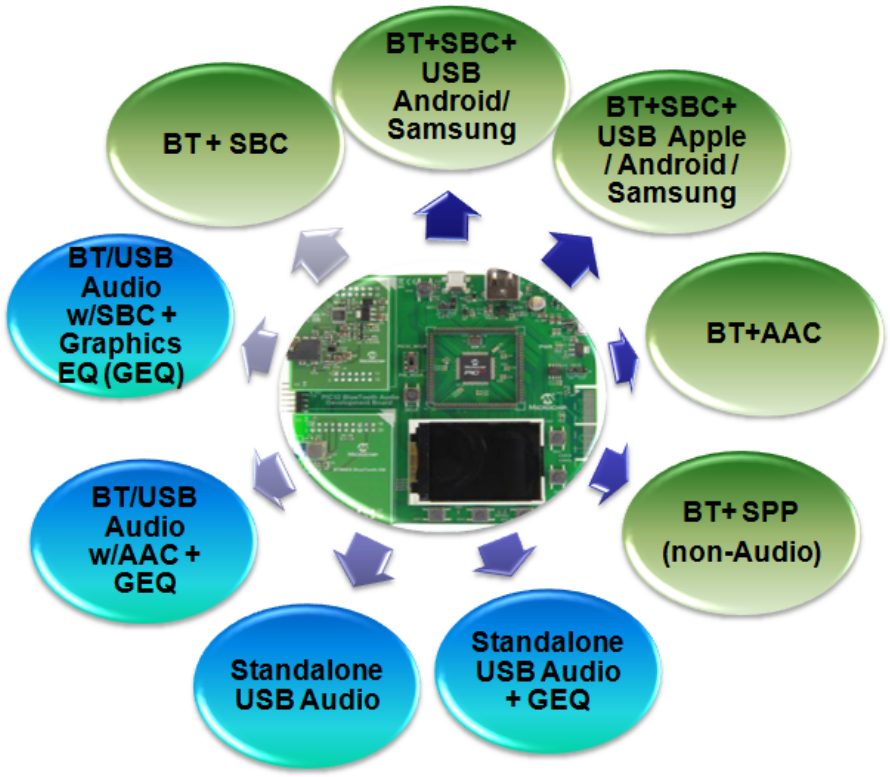
Configuring the Library

This topic provides information on configuring the library.

Description

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

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



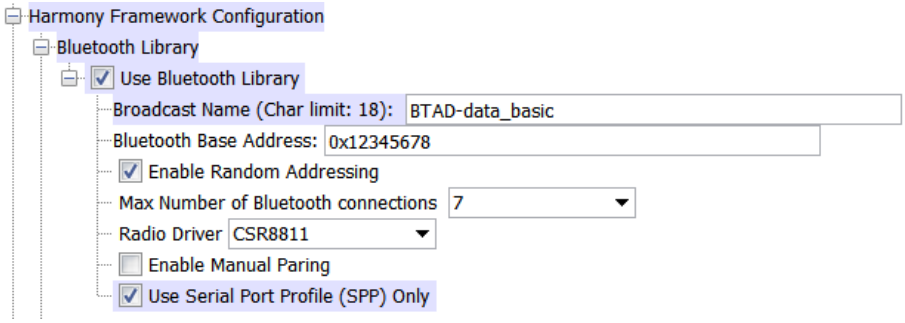
Configuring the MHC

This section provides information on configuring the MPLAB Harmony Configurator (MHC) for use with the Bluetooth Library.

Description

The following figure shows a typical MHC configuration. Selecting "Use Bluetooth Library" in MHC adds the Searan Dotstack Bluetooth Library to the project and enables a menu of selections used to configure the application, as follows.

- A text string "Broadcast Name (Char limit: 18) - The name of the device used to advertise to other nearby devices. This defines the BT_CONNECTION_NAME macro.
- The "Bluetooth Base Address" must be an 8-digit Hex value (C-style, as shown), representing the first eight digits of the Bluetooth MAC address. The complete MAC address is suffixed by four more digits, which are generated either as random digits (when "Enable Random Addressing" is selected, as shown) or manually entered in the "Bluetooth Suffix Address" (when "Enable Random Addressing" is not selected).
- "Max Number of Bluetooth Connections", gives the maximum number of SPP connections
- "Radio Driver" select the desired type from the menu:
- CSR8811 - for the CSR88xx/BTM805 Radio module
- RDA Radio
- CC2564/PAN13xx Radio module
- IS1662/MFR4662 Radio module
- "Enable Manual Pairing" forces the Bluetooth Stack to always request a manual pairing sequence using a pin code, rather than automatic pairing
- Selecting "Use Serial Port Profile (SPP)" and/or "Use A2DP/AVRCP Profile" indicates which version of the library to use to support the selected profile(s)



The following macros are generated to `system_config.h`:

- BT_DEVICE_DESIGN_ID - defined as the Bluetooth Base Address

- BT_DEVICE_ID_2LSB_RANDOMIZE - Boolean indicating a random 4-digit suffix
- BT_DEVICE_ID_2LSB - manual entry of the suffix
- BT_CONTROLLER - indicates the radio module being used
- BT_MAX_SPP_PORTS - provides the maximum number of SPP connections

Bluetooth Stack Library Configuration Profile Macros

AVCTP Configuration Macros

	Name	Description
	__AVCTP_CONFIG_H	This is macro __AVCTP_CONFIG_H.

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

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

AVDTP Configuration Macros

	Name	Description
	AVDTP_ALLOCATE_BUFFERS_FUNCTION	This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

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

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

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

AVRCP Configuration Macros

	Name	Description
	__AVRCP_CONFIG_H	This is macro __AVRCP_CONFIG_H.

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

	AVRCP_ALLOCATE_BUFFERS_VARS	This is macro AVRCP_ALLOCATE_BUFFERS_VARS.
	AVRCP_CMD_TIMEOUT	brief Command timeout ingroup avrcp_config details This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.
	AVRCP_MAX_CHANNELS	brief Maximum number of remote devices a local device can be connected to ingroup avrcp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.
	AVRCP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avrcp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS * AVRCP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.

	AVRCP_MAX_CMD_PARAM_LEN	brief Maximum length of command parameters ingroup avrcp_config details This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.
	AVRCP_MAX_DEVICE_NAME_LEN	brief Maximum length of device name ingroup avrcp_config details This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.
	AVRCP_MAX_SEARCH_RESULTS	brief Maximum number of devices to find ingroup avrcp_config details This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

Bluetooth Support Configuration Macros

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

HCI Configuration Macros

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

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

L2CAP Configuration Macros

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

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

RFCOMM Configuration Macros

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

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

SDP Configuration Macros

	Name	Description

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

	SDP_ALLOCATE_BUFFERS_VARS	This is macro SDP_ALLOCATE_BUFFERS_VARS.

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

SPP Configuration Macros

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

Description

This section provides the configuration profile macros for the Bluetooth Stack Library.

Bluetooth Support Configuration Macros

ATT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
ATT_ALLOCATE_BUFFERS();
```

Description

This is function ATT_ALLOCATE_BUFFERS.

ATT_CLIENT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
ATT_CLIENT_ALLOCATE_BUFFERS();
```

Description

This is function ATT_CLIENT_ALLOCATE_BUFFERS.

FTP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
FTP_ALLOCATE_BUFFERS();
```

Description

This is function FTP_ALLOCATE_BUFFERS.

GATT_CLIENT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
GATT_CLIENT_ALLOCATE_BUFFERS();
```

Description

This is function GATT_CLIENT_ALLOCATE_BUFFERS.

HCI_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCI_ALLOCATE_BUFFERS();
```

Description

This is function HCI_ALLOCATE_BUFFERS.

HFP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HFP_ALLOCATE_BUFFERS();
```

Description

This is function HFP_ALLOCATE_BUFFERS.

HID_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HID_ALLOCATE_BUFFERS();
```

Description

This is function HID_ALLOCATE_BUFFERS.

HSP_AG_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HSP_AG_ALLOCATE_BUFFERS();
```

Description

This is function HSP_AG_ALLOCATE_BUFFERS.

HSP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HSP_ALLOCATE_BUFFERS();
```

Description

This is function HSP_ALLOCATE_BUFFERS.

IAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP_ALLOCATE_BUFFERS();
```

Description

This is function IAP_ALLOCATE_BUFFERS.

IAP_BT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP_BT_ALLOCATE_BUFFERS();
```

Description

This is function IAP_BT_ALLOCATE_BUFFERS.

IAP2_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP2_ALLOCATE_BUFFERS();
```

Description

This is function IAP2_ALLOCATE_BUFFERS.

L2CAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
L2CAP_ALLOCATE_BUFFERS();
```

Description

This is function L2CAP_ALLOCATE_BUFFERS.

OBEX_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
OBEX_ALLOCATE_BUFFERS();
```

Description

This is function OBEX_ALLOCATE_BUFFERS.

PBAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
PBAP_ALLOCATE_BUFFERS();
```

Description

This is function PBAP_ALLOCATE_BUFFERS.

RFCOMM_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
RFCOMM_ALLOCATE_BUFFERS();
```

Description

This is function RFCOMM_ALLOCATE_BUFFERS.

SDP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SDP_ALLOCATE_BUFFERS();
```

Description

This is function SDP_ALLOCATE_BUFFERS.

SMP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SMP_ALLOCATE_BUFFERS();
```

Description

This is function SMP_ALLOCATE_BUFFERS.

SPP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SPP_ALLOCATE_BUFFERS();
```

Description

This is function SPP_ALLOCATE_BUFFERS.

BT_ENABLE_BLE Macro

File

bt_oem_config.h

C

```
#define BT_ENABLE_BLE
```

Description

This is macro BT_ENABLE_BLE.

BT_INCLUDE_RFCOMM Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_RFCOMM
```

Description

This is macro BT_INCLUDE_RFCOMM.

BT_LOG_LEVEL_MAX Macro

File

bt_oem_config.h

C

```
#define BT_LOG_LEVEL_MAX BT_LOG_LEVEL_OFF
```

Description

This is macro BT_LOG_LEVEL_MAX.

BT_LOG_LEVEL_MIN Macro

File

bt_oem_config.h

C

```
#define BT_LOG_LEVEL_MIN BT_LOG_LEVEL
```

Description

This is macro BT_LOG_LEVEL_MIN.

__BT_APP_CONFIG_H Macro

File

bt_oem_config.h

C

```
#define __BT_APP_CONFIG_H
```

Description

This is macro __BT_APP_CONFIG_H.

BT_INCLUDE_IAP Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_IAP
```

Description

This is macro BT_INCLUDE_IAP.

BT_INCLUDE_IAP2 Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_IAP2
```

Description

This is macro BT_INCLUDE_IAP2.

IAP_BT_MAX_TRANSPORTS Macro

File

bt_oem_config.h

C

```
#define IAP_BT_MAX_TRANSPORTS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP_BT_MAX_TRANSPORTS.

IAP_MAX_SESSIONS Macro

File

bt_oem_config.h

C

```
#define IAP_MAX_SESSIONS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP_MAX_SESSIONS.

IAP_RX_BUFFER_SIZE Macro

File

bt_oem_config.h

C

```
#define IAP_RX_BUFFER_SIZE IAPEA_MAX_PACKET_SIZE
```

Description

This is macro IAP_RX_BUFFER_SIZE.

IAP2_MAX_PACKET_SIZE Macro

File

bt_oem_config.h

C

```
#define IAP2_MAX_PACKET_SIZE IAPEA_MAX_PACKET_SIZE
```

Description

This is macro IAP2_MAX_PACKET_SIZE.

IAP2_MAX_SESSIONS Macro

File

bt_oem_config.h

C

```
#define IAP2_MAX_SESSIONS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP2_MAX_SESSIONS.

AVCTP Configuration Macros

__AVCTP_CONFIG_H Macro

File

avctp_config.h

C

```
#define __AVCTP_CONFIG_H
```

Description

This is macro __AVCTP_CONFIG_H.

AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avctp_config.h

C

```
#define AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avctp_config Configuration
- ingroup avctp

*

- This module describes parameters used to configure AVCTP layer.

*

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

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

// HCI, L2CAP and SDP must always be present

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

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

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

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

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

#include "cdbt/bt/bt_oem_config.h"

endcode

AVCTP_ALLOCATE_BUFFERS_VARS Macro

File

avctp_config.h

C

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

Description

brief Maximum number of message buffers ingroup avctp_config

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

AVCTP_MAX_CHANNELS Macro

File

avctp_config.h

C

```
#define AVCTP_MAX_CHANNELS
```

Description

brief Maximum number of AVCTP channels ingroup avctp_config

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

AVCTP_MAX_TRANSPORT_CHANNELS Macro

File

avctp_config.h

C

```
#define AVCTP_MAX_TRANSPORT_CHANNELS
```

Description

brief Maximum number of AVCTP transports ingroup avctp_config

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

AVDTP Configuration Macros

AVDTP_ALLOCATE_BUFFERS_FUNCTION Macro

File

avdtp_config.h

C

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

Description

This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avdtp_config.h

C

```
#define AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avdtp_config Configuration
- ingroup avdtp

*

- This module describes parameters used to configure AVDTP layer.

*

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

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

// HCI, L2CAP and SDP must always be present

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

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

L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...

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

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

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

#include "cdbt/bt/bt_oem_config.h"

endcode

AVDTP_ALLOCATE_BUFFERS_VARS Macro

File

avdtp_config.h

C

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

Description

This is macro AVDTP_ALLOCATE_BUFFERS_VARS.

AVDTP_CMD_ABORT Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_ABORT 10
```

Description

This is macro AVDTP_CMD_ABORT.

AVDTP_CMD_CLOSE Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_CLOSE 8
```

Description

This is macro AVDTP_CMD_CLOSE.

AVDTP_CMD_DISCOVER Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_DISCOVER 1
```

Description

This is macro AVDTP_CMD_DISCOVER.

AVDTP_CMD_GET_CAPABILITIES Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_CAPABILITIES 2
```

Description

This is macro AVDTP_CMD_GET_CAPABILITIES.

AVDTP_CMD_GET_CONFIGURATION Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_CONFIGURATION 4
```

Description

This is macro AVDTP_CMD_GET_CONFIGURATION.

AVDTP_CMD_OPEN Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_OPEN 6
```

Description

This is macro AVDTP_CMD_OPEN.

AVDTP_CMD_RECONFIGURE Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_RECONFIGURE 5
```

Description

This is macro AVDTP_CMD_RECONFIGURE.

AVDTP_CMD_SECURITY_CONTROL Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SECURITY_CONTROL 11
```

Description

This is macro AVDTP_CMD_SECURITY_CONTROL.

AVDTP_CMD_SET_CONFIGURATION Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SET_CONFIGURATION 3
```

Description

This is macro AVDTP_CMD_SET_CONFIGURATION.

AVDTP_CMD_START Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_START 7
```

Description

This is macro AVDTP_CMD_START.

AVDTP_CMD_SUSPEND Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SUSPEND 9
```

Description

This is macro AVDTP_CMD_SUSPEND.

AVDTP_CODEC_CONFIG_BUFFER_LEN Macro

File

avdtp_config.h

C

```
#define AVDTP_CODEC_CONFIG_BUFFER_LEN 16
```

Description

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

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

AVDTP_MAX_CMD_BUFFERS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_CMD_BUFFERS
```

Description

brief Maximum number of command buffers. ingroup avdtp_config

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

AVDTP_MAX_CMD_PARAM_LEN Macro

File

avdtp_config.h

C

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

Description

brief Maximum length of control command parameters ingroup avdtp_config

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

AVDTP_MAX_REMOTE_DEVICES Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_REMOTE_DEVICES
```

Description

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

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

AVDTP_MAX_SEP Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_SEP
```

Description

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

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

AVDTP_MAX_STREAMS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_STREAMS
```

Description

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

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

AVDTP_MAX_TRANSPORT_CHANNELS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_TRANSPORT_CHANNELS
```

Description

brief Maximum number of transport channels. ingroup avdtp_config

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

AVDTP_MAX_TX_BUFFER_LEN Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_TX_BUFFER_LEN 32
```

Description

brief Size of the transmit buffer. ingroup avdtp_config

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

AVRCP Configuration Macros

__AVRCP_CONFIG_H Macro

File

avrcp_config.h

C

```
#define __AVRCP_CONFIG_H
```

Description

This is macro __AVRCP_CONFIG_H.

AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avrcp_config.h

C

```
#define AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avrcp_config Configuration
- ingroup avrcp

*

- This module describes parameters used to configure AVRCP layer.

*

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

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

// HCI, L2CAP and SDP must always be present

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

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

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

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

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

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

#include "cdbt/bt/bt_oem_config.h"

endcode

AVRCP_ALLOCATE_BUFFERS_VARS Macro

File

avrcp_config.h

C

```
#define AVRCP_ALLOCATE_BUFFERS_VARS \
    bt_avrcp_channel_t  _avrcp_channels[AVRCP_MAX_CHANNELS];    \
    const bt_byte       _avrcp_max_channels = AVRCP_MAX_CHANNELS; \
    \
    bt_buffer_header_t  _avrcp_cmd_buffer_headers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CHANNELS)];    \
    bt_av_command_t     _avrcp_cmd_buffers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CHANNELS)];    \
```

```
    const bt_byte       _avrcp_max_cmd_buffers = AVRCP_MAX_CMD_BUFFERS; \
    bt_byte             _avrcp_cmd_param_buffers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CMD_PARAM_LEN) *
(AVRCP_MAX_CHANNELS)];    \
    const bt_int        _avrcp_max_cmd_param_len = AVRCP_MAX_CMD_PARAM_LEN;    \
    const bt_byte       _avrcp_max_search_results = AVRCP_MAX_SEARCH_RESULTS;    \
    bt_avrcp_device_t   _avrcp_devices_buffer[AVRCP_MAX_SEARCH_RESULTS];    \
    const bt_byte       _avrcp_max_device_name_len = AVRCP_MAX_DEVICE_NAME_LEN;    \
    bt_byte             _avrcp_device_name_buffers[(AVRCP_MAX_SEARCH_RESULTS) * (AVRCP_MAX_DEVICE_NAME_LEN
+ 1)];    \
    const bt_byte       _avrcp_cmd_timeout = (AVRCP_CMD_TIMEOUT) / 100;    \
    \
    void (*_bt_avrcp_response_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_sent_t* evt_param) =
AVRCP_RESPONSE_SENT_HANDLER;    \
    void (*_bt_avrcp_command_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_received_t* evt_param) =
AVRCP_COMMAND_HANDLER;    \
    void (*_bt_avrcp_command_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_sent_t* evt_param) =
AVRCP_COMMAND_SENT_HANDLER;    \
    void (*_bt_avrcp_response_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_received_t* evt_param) =
AVRCP_RESPONSE_HANDLER;    \
    \
    AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro AVRCP_ALLOCATE_BUFFERS_VARS.

AVRCP_CMD_TIMEOUT Macro

File

avrcp_config.h

C

```
#define AVRCP_CMD_TIMEOUT 10000
```

Description

brief Command timeout ingroup avrcp_config

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

AVRCP_MAX_CHANNELS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CHANNELS
```

Description

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

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

AVRCP_MAX_CMD_BUFFERS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CMD_BUFFERS 1
```

Description

brief Maximum number of command buffers. ingroup avrcp_config

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

AVRCP_MAX_CMD_PARAM_LEN Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CMD_PARAM_LEN 512
```

Description

brief Maximum length of command parameters ingroup avrcp_config

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

AVRCP_MAX_DEVICE_NAME_LEN Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_DEVICE_NAME_LEN 20
```

Description

brief Maximum length of device name ingroup avrcp_config

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

AVRCP_MAX_SEARCH_RESULTS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_SEARCH_RESULTS 7
```

Description

brief Maximum number of devices to find ingroup avrcp_config

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

HCI Configuration Macros

BT_ENABLE_SCO Macro

File

hci_config.h

C

```
#define BT_ENABLE_SCO BT_FALSE
```

Description

This is macro BT_ENABLE_SCO.

BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER.

BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER.

BT_HCI_EVT_CONNECTION_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_CONNECTION_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_CONNECTION_REQUEST_HANDLER.

BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER bt_hci_evt_encryption_change_handler
```

Description

This is macro BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER.

BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER.

BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER.

BT_HCI_EVT_INQUIRY_RESULT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_RESULT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_RESULT_HANDLER.

BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER.

BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER.

BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER.

BT_HCI_EVT_MODE_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_MODE_CHANGE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_MODE_CHANGE_HANDLER.

BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER.

BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER.

BT_HCI_EVT_ROLE_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_ROLE_CHANGE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_ROLE_CHANGE_HANDLER.

BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER.

BT_LE_EVT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_LE_EVT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_LE_EVT_HANDLER.

BT_SSP_EVT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_SSP_EVT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_SSP_EVT_HANDLER.

HCI_ALLOCATE_BUFFERS_FUNCTION Macro

File

hci_config.h

C

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

Description

This is macro HCI_ALLOCATE_BUFFERS_FUNCTION.

HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

hci_config.h

C

```
#define HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

HCI_ALLOCATE_BUFFERS_VARS Macro

File

hci_config.h

C

```
#define HCI_ALLOCATE_BUFFERS_VARS \
    bt_buffer_header_t  _hci_cmd_buffer_headers[HCI_MAX_CMD_BUFFERS];    \
    bt_hci_command_t    _hci_cmd_buffers[HCI_MAX_CMD_BUFFERS];    \
    const bt_byte       _hci_max_cmd_buffers = HCI_MAX_CMD_BUFFERS; \
    bt_byte             _hci_cmd_param_buffers[HCI_MAX_CMD_BUFFERS * HCI_MAX_CMD_PARAM_LEN];    \
    const bt_byte       _hci_max_cmd_param_len = HCI_MAX_CMD_PARAM_LEN;    \
    \
    bt_buffer_header_t  _hci_send_data_buffer_headers[HCI_MAX_DATA_BUFFERS];    \
    bt_hci_data_t       _hci_send_data_buffers[HCI_MAX_DATA_BUFFERS];    \
```

```
const bt_byte       _hci_max_data_buffers = HCI_MAX_DATA_BUFFERS; \
\
bt_hci_conn_state_t _hci_connections[HCI_MAX_HCI_CONNECTIONS];    \
const bt_byte       _hci_max_hci_connections = HCI_MAX_HCI_CONNECTIONS; \
\
bt_byte             _recv_buffer[HCI_RX_BUFFER_LEN];    \
bt_byte             _send_buffer[HCI_TX_BUFFER_LEN];    \
const bt_uint       _hci_rcv_buffer_len = HCI_RX_BUFFER_LEN;    \
const bt_uint       _hci_tx_buffer_len = HCI_TX_BUFFER_LEN;    \
\
bt_byte             _conn_state_recv_buffers[(HCI_L2CAP_BUFFER_LEN) * (HCI_MAX_HCI_CONNECTIONS)];    \
const bt_uint       _hci_l2cap_buffer_len = HCI_L2CAP_BUFFER_LEN;    \
\
const bt_bool       _hci_enable_ctrl_to_host_flow_control = HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL;    \
\
const bt_byte       _hci_max_connect_attempts = HCI_MAX_CONNECT_ATTEMPTS;    \
\
const bt_bool       _hci_enable_sco = BT_ENABLE_SCO;    \
void (*_hci_recv_sco_data_packet_fp)(bt_byte* pbuf);    \
\
void (*_bt_ssp_init)(void);    \
void (*_bt_ssp_evt_handler)(bt_hci_event_t* evt);    \
\
void (*_bt_hci_le_init)(bt_hci_le_ctrl_state_t* le_ctrl_state);    \
HCI_DECLARE_LE_CONN_STATES    \
HCI_DECLARE_LE_CTRL_STATE    \
\
HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro HCI_ALLOCATE_BUFFERS_VARS.

HCI_DECLARE_LE_CONN_STATES Macro

File

hci_config.h

C

```
#define HCI_DECLARE_LE_CONN_STATES
```

Description

This is macro HCI_DECLARE_LE_CONN_STATES.

HCI_DECLARE_LE_CTRL_STATE Macro

File

hci_config.h

C

```
#define HCI_DECLARE_LE_CTRL_STATE
```

Description

This is macro HCI_DECLARE_LE_CTRL_STATE.

HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL Macro

File

hci_config.h

C

```
#define HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL BT_FALSE
```

Description

This is macro HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL.

HCI_INIT_LE_CONN_STATES Macro

File

hci_config.h

C

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

Description

This is macro HCI_INIT_LE_CONN_STATES.

HCI_INIT_LE_CTRL_STATE Macro

File

hci_config.h

C

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

Description

This is macro HCI_INIT_LE_CTRL_STATE.

HCI_INIT_SCO_HANDLERS Macro

File

hci_config.h

C

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

Description

This is macro HCI_INIT_SCO_HANDLERS.

HCI_INIT_SSP_HANDLERS Macro

File

hci_config.h

C

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

Description

This is macro HCI_INIT_SSP_HANDLERS.

HCI_L2CAP_BUFFER_LEN Macro

File

hci_config.h

C

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

Description

This is macro HCI_L2CAP_BUFFER_LEN.

HCI_SIZEOF_LE_CONN_STATES Macro

File

hci_config.h

C

```
#define HCI_SIZEOF_LE_CONN_STATES 0
```

Description

This is macro HCI_SIZEOF_LE_CONN_STATES.

HCI_SIZEOF_LE_CTRL_STATE Macro

File

hci_config.h

C

```
#define HCI_SIZEOF_LE_CTRL_STATE 0
```

Description

This is macro HCI_SIZEOF_LE_CTRL_STATE.

HCI_TX_BUFFER_LEN Macro

File

hci_config.h

C

```
#define HCI_TX_BUFFER_LEN HCI_RX_BUFFER_LEN
```

Description

This is macro HCI_TX_BUFFER_LEN.

__HCI_CONFIG_EVENT_HANDLERS_H Macro

File

hci_config_event_handlers.h

C

```
#define __HCI_CONFIG_EVENT_HANDLERS_H
```

Description

This is macro __HCI_CONFIG_EVENT_HANDLERS_H.

__HCI_CONFIG_H Macro

File

hci_config.h

C

```
#define __HCI_CONFIG_H
```

Description

This is macro __HCI_CONFIG_H.

HCI_MAX_CONNECT_ATTEMPTS Macro

File

hci_config.h

C

```
#define HCI_MAX_CONNECT_ATTEMPTS 4
```

Description

This is macro HCI_MAX_CONNECT_ATTEMPTS.

L2CAP Configuration Macros

L2CAP_ALLOCATE_BUFFERS_FUNCTION Macro

File

l2cap_config.h

C

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

Description

This is macro L2CAP_ALLOCATE_BUFFERS_FUNCTION.

L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

l2cap_config.h

C

```
#define L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

L2CAP_ALLOCATE_BUFFERS_VARS Macro

File

l2cap_config.h

C

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

Description

This is macro L2CAP_ALLOCATE_BUFFERS_VARS.

L2CAP_FIXED_CHANNELS_DECL Macro

File

l2cap_config.h

C

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

Description

This is macro L2CAP_FIXED_CHANNELS_DECL.

L2CAP_HCI_PACKET_TYPE Macro

File

l2cap_config.h

C

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

Description

- brief L2CAP_HCI_PACKET_TYPE.
- ingroup btconfig

*

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

enable all packet types

L2CAP_HCI_PAGE_SCAN_REPETITION_MODE Macro

File

l2cap_config.h

C

```
#define L2CAP_HCI_PAGE_SCAN_REPETITION_MODE HCI_PAGE_SCAN_REPETITION_MODE_R0
```

Description

brief L2CAP_HCI_PAGE_SCAN_REPETITION_MODE. ingroup btconfig

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

The default value is HCI_PAGE_SCAN_REPETITION_MODE_R0.

L2CAP_HCI_ROLE_SWITCH Macro

File

l2cap_config.h

C

```
#define L2CAP_HCI_ROLE_SWITCH HCI_ROLE_SWITCH_ALLOW
```

Description

brief L2CAP_HCI_ROLE_SWITCH. ingroup btconfig

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

L2CAP_IDLE_CONNECTION_TIMEOUT Macro

File

l2cap_config.h

C

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

Description

seconds

PACK_CONFIG_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONFIG_REQUEST NULL
```

Description

This is macro PACK_CONFIG_REQUEST.

PACK_CONFIG_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONFIG_RESPONSE NULL
```

Description

This is macro PACK_CONFIG_RESPONSE.

PACK_CONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONN_REQUEST NULL
```

Description

This is macro PACK_CONN_REQUEST.

PACK_CONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONN_RESPONSE NULL
```

Description

This is macro PACK_CONN_RESPONSE.

PACK_DCONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_DCONN_REQUEST NULL
```

Description

This is macro PACK_DCONN_REQUEST.

PACK_DCONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_DCONN_RESPONSE NULL
```

Description

This is macro PACK_DCONN_RESPONSE.

PACK_INFO_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_INFO_REQUEST NULL
```

Description

This is macro PACK_INFO_REQUEST.

PACK_INFO_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_INFO_RESPONSE NULL
```

Description

This is macro PACK_INFO_RESPONSE.

PROCESS_CONFIG_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONFIG_REQ _process_unknown_req
```

Description

This is macro PROCESS_CONFIG_REQ.

PROCESS_CONFIG_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONFIG_RES _process_unknown_res
```

Description

This is macro PROCESS_CONFIG_RES.

PROCESS_CONN_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONN_REQ _process_unknown_req
```

Description

This is macro PROCESS_CONN_REQ.

PROCESS_CONN_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONN_RES _process_unknown_res
```

Description

This is macro PROCESS_CONN_RES.

PROCESS_DCONN_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_DCONN_REQ _process_unknown_req
```

Description

This is macro PROCESS_DCONN_REQ.

PROCESS_DCONN_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_DCONN_RES _process_unknown_res
```

Description

This is macro PROCESS_DCONN_RES.

PROCESS_INFO_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_INFO_REQ _process_unknown_req
```

Description

This is macro PROCESS_INFO_REQ.

PROCESS_INFO_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_INFO_RES _process_unknown_res
```

Description

This is macro PROCESS_INFO_RES.

READ_CONFIG_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONFIG_REQUEST NULL
```

Description

This is macro READ_CONFIG_REQUEST.

READ_CONFIG_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONFIG_RESPONSE NULL
```

Description

This is macro READ_CONFIG_RESPONSE.

READ_CONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONN_REQUEST NULL
```

Description

This is macro READ_CONN_REQUEST.

READ_CONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONN_RESPONSE NULL
```

Description

This is macro READ_CONN_RESPONSE.

READ_DCONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_DCONN_REQUEST NULL
```

Description

This is macro READ_DCONN_REQUEST.

READ_DCONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_DCONN_RESPONSE NULL
```

Description

This is macro READ_DCONN_RESPONSE.

READ_INFO_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_INFO_REQUEST NULL
```

Description

This is macro READ_INFO_REQUEST.

READ_INFO_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_INFO_RESPONSE NULL
```

Description

This is macro READ_INFO_RESPONSE.

__L2CAP_CONFIG_H Macro

File

l2cap_config.h

C

```
#define __L2CAP_CONFIG_H
```

Description

This is macro __L2CAP_CONFIG_H.

__L2CAP_CONFIG_HANDLERS_H Macro

File

l2cap_config_handlers.h

C

```
#define __L2CAP_CONFIG_HANDLERS_H
```

Description

This is macro __L2CAP_CONFIG_HANDLERS_H.

L2CAP_DECL_ERETR_FUNCTIONS Macro

File

l2cap_config.h

C

```
#define L2CAP_DECL_ERETR_FUNCTIONS \
        bt_l2cap_channel_ext_t*  _l2cap_channels_ext = NULL;    \
        \
        void (*_l2cap_eretr_recv_fp)(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte* pdata, bt_int
len) = NULL;    \
        bt_bool (*_l2cap_eretr_send_data_fp)(bt_l2cap_channel_t* pch, bt_byte* data, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param) = NULL;    \
        bt_bool (*_l2cap_eretr_send_smart_data_fp)(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int
len, bt_l2cap_send_data_callback_fp cb, void* cb_param) = NULL;    \
        bt_bool (*_l2cap_eretr_handle_xmit_event_fp)(bt_l2cap_xmit_event_param_t* param) = NULL;    \
        void (*_l2cap_eretr_pack_config_request_fp)(bt_l2cap_channel_t* channel, bt_byte* buffer, bt_int
buffer_len, bt_int* offset) = NULL;
```

Description

This is macro L2CAP_DECL_ERETR_FUNCTIONS.

L2CAP_MAX_FIXED_CHANNELS Macro

File

l2cap_config.h

C

```
#define L2CAP_MAX_FIXED_CHANNELS 0
```

Description

This is macro L2CAP_MAX_FIXED_CHANNELS.

RFCOMM Configuration Macros

RFCOMM_ALLOCATE_BUFFERS_FUNCTION Macro

File

rfcomm_config.h

C

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

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_FUNCTION.

RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

RFCOMM_ALLOCATE_BUFFERS_VARS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ALLOCATE_BUFFERS_VARS \
    bt_rfcomm_session_t _rfcomm_sessions[RFCOMM_MAX_SESSIONS];    \
    const bt_byte       _rfcomm_max_sessions = RFCOMM_MAX_SESSIONS;    \
    bt_rfcomm_dlc_t     _rfcomm_dlcs[(RFCOMM_MAX_DLCS) * (RFCOMM_MAX_SESSIONS)];    \
    const bt_byte       _rfcomm_max_dlcs = RFCOMM_MAX_DLCS;    \
    bt_rfcomm_server_channel_t _rfcomm_channels[(RFCOMM_MAX_SERVER_CHANNELS) * (RFCOMM_MAX_SESSIONS)];    \
    const bt_byte       _rfcomm_max_channels = RFCOMM_MAX_SERVER_CHANNELS;    \
    const bt_uint       _rfcomm_pdu_size = (RFCOMM_INFO_LEN) + (RFCOMM_FRAME_HEADER_LEN);    \
    const bt_uint       _rfcomm_info_len = RFCOMM_INFO_LEN;    \
    bt_buffer_header_t  _rfcomm_cmd_buffer_headers[RFCOMM_MAX_CMD_BUFFERS];    \
    bt_rfcomm_command_t _rfcomm_cmd_buffers[RFCOMM_MAX_CMD_BUFFERS];    \
    const bt_byte       _rfcomm_max_cmd_buffers = RFCOMM_MAX_CMD_BUFFERS;    \
    const bt_byte       _rfcomm_local_credit = RFCOMM_LOCAL_CREDIT;    \
    const bt_bool       _rfcomm_enable_multidevice_channels = RFCOMM_ENABLE_MULTIDEVICE_CHANNELS;    \
    RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL    \
    RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_VARS.

RFCOMM_BUFFER_SIZE Macro

File

rfcomm_config.h

C

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

Description

This is macro RFCOMM_BUFFER_SIZE.

RFCOMM_ENABLE_MULTIDEVICE_CHANNELS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ENABLE_MULTIDEVICE_CHANNELS BT_TRUE
```

Description

brief Enable multi-device server channels. ingroup rfcomm_config

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

RFCOMM_INFO_LEN Macro

File

rfcomm_config.h

C

```
#define RFCOMM_INFO_LEN
```

Description

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

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

RFCOMM_LOCAL_CREDIT Macro

File

rfcomm_config.h

C

```
#define RFCOMM_LOCAL_CREDIT
```

Description

brief The number of receive buffers. ingroup rfcomm_config

details This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory allocation is done in SPP layer.

RFCOMM_MAX_CMD_BUFFERS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_CMD_BUFFERS
```

Description

brief Maximum number of command buffers. ingroup rfcomm_config

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

RFCOMM_MAX_DLCS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_DLCS
```

Description

brief Maximum number of DLCs ingroup rfcomm_config

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

RFCOMM_MAX_SERVER_CHANNELS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_SERVER_CHANNELS
```

Description

brief Maximum number of Server channels ingroup rfcomm_config

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

RFCOMM_MAX_SESSIONS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_SESSIONS
```

Description

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

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

__RFCOMM_CONFIG_H Macro

File

rfcomm_config.h

C

```
#define __RFCOMM_CONFIG_H
```

Description

This is macro __RFCOMM_CONFIG_H.

RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL Macro

File

rfcomm_config.h

C

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

Description

This is macro RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL.

SDP Configuration Macros

SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

sdp_config.h

C

```
#define SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup sdp_config Configuration
- ingroup sdp

*

- This module describes parameters used to configure SDP.

*

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

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

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

// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define

HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...

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

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

#include "cdbt/bt/bt_oem_config.h"

endcode

SDP_ALLOCATE_BUFFERS_VARS Macro

File

sdp_config.h

C

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

Description

This is macro SDP_ALLOCATE_BUFFERS_VARS.

SDP_MAX_ATTRIBUTE_RESULT_LEN Macro

File

sdp_config.h

C

```
#define SDP_MAX_ATTRIBUTE_RESULT_LEN
```

Description

brief Maximum number of attributes to find ingroup sdp_config

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

SDP_MAX_PDU_BUFFERS Macro

File

sdp_config.h

C

```
#define SDP_MAX_PDU_BUFFERS
```

Description

brief Maximum number of SDP server PDU buffers. ingroup sdp_config

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

SDP_MAX_SEARCH_RESULT_LEN Macro

File

sdp_config.h

C

```
#define SDP_MAX_SEARCH_RESULT_LEN
```

Description

brief Maximum number of service records to find. ingroup sdp_config

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

__SDP_CONFIG_H Macro

File

sdp_config.h

C

```
#define __SDP_CONFIG_H
```

Description

This is macro __SDP_CONFIG_H.

SPP Configuration Macros

SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

spp_config.h

C

```
#define SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

SPP_ALLOCATE_BUFFERS_VARS Macro

File

spp_config.h

C

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

Description

This is macro SPP_ALLOCATE_BUFFERS_VARS.

SPP_DECLARE_FRAME_BUFFERS Macro

File

spp_config.h

C

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

Description

This is macro SPP_DECLARE_FRAME_BUFFERS.

SPP_DISABLE_BUFFERING Macro

File

spp_config.h

C

```
#define SPP_DISABLE_BUFFERING 1
```

Description

This is macro SPP_DISABLE_BUFFERING.

SPP_FRAME_BUFFERS_RAM_SIZE Macro

File

spp_config.h

C

```
#define SPP_FRAME_BUFFERS_RAM_SIZE 0
```

Description

This is macro SPP_FRAME_BUFFERS_RAM_SIZE.

SPP_FRAME_BUFFERS_SIZE Macro

File

spp_config.h

C

```
#define SPP_FRAME_BUFFERS_SIZE
```

Description

This is macro SPP_FRAME_BUFFERS_SIZE.

SPP_MAX_PORTS Macro

File

spp_config.h

C

```
#define SPP_MAX_PORTS
```

Description

brief Maximum number of SPP ports. ingroup spp_config

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

RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS * RFCOMM_MAX_SESSIONS.

__SPP_CONFIG_H Macro

File

spp_config.h

C

```
#define __SPP_CONFIG_H
```

Description

This is macro __SPP_CONFIG_H.

Building the Library

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

Library Interface

A2DP Data Types and Constants

	Name	Description
	_bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	_bt_a2dp_event_u	brief Parameter to an application callback. ingroup a2dp details This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_a2dp_evt_open_and_start_stream_completed_s	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	_bt_a2dp_mgr_t	brief A2DP manager. ingroup a2dp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with ::bt_a2dp_get_mgr.
	_bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	_bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	_bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	bt_a2dp_event_t	This is type bt_a2dp_event_t.
	bt_a2dp_evt_open_and_start_stream_completed_t	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	bt_a2dp_find_server_callback_fp	brief Notify the application of the result of searching for a remote A2DP entity (sourse or sink) ingroup a2dp details This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed. param supported_features Features supported by a remote A2DP entity. param found c TRUE if an A2DP entity has been found on the remote device. c FALSE otherwise. param param pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its c callback_param parameter.

	bt_a2dp_mgr_callback_fp	brief A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr A2DP manager. param evt A2DP event. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local... more
	bt_a2dp_mgr_t	This is type bt_a2dp_mgr_t.
	bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	__A2DP_CODEC_AAC_H	This is macro __A2DP_CODEC_AAC_H.
	__A2DP_CODEC_MPEG_H	This is macro __A2DP_CODEC_MPEG_H.
	__A2DP_CODEC_SBC_H	This is macro __A2DP_CODEC_SBC_H.
	__A2DP_H	This is macro __A2DP_H.
	__A2DP_PRIVATE_H	This is macro __A2DP_PRIVATE_H.
	A2DP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	A2DP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	A2DP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	A2DP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	A2DP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	A2DP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	A2DP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	A2DP_EVT_DISCOVER_SEP_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	A2DP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	A2DP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	A2DP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	A2DP_EVT_NOTHING	This is macro A2DP_EVT_NOTHING.
	A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED	< This event is generated when a local device completed "open and start" request.
	A2DP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	A2DP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	A2DP_EVT_RECONFIGURE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

	A2DP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	A2DP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	A2DP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	A2DP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	A2DP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	A2DP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	A2DP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	A2DP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.
	A2DP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.
	A2DP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	A2DP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	A2DP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	A2DP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	A2DP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	A2DP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	A2DP_MANAGER_STATE_CONNECTING	This is macro A2DP_MANAGER_STATE_CONNECTING.
	A2DP_MANAGER_STATE_IDLE	This is macro A2DP_MANAGER_STATE_IDLE.
	A2DP_SINK_FEATURE_AMPLIFIER	< Amplifier
	A2DP_SINK_FEATURE_HEADPHONE	< Headphone
	A2DP_SINK_FEATURE_RECORDER	< Recorder
	A2DP_SINK_FEATURE_SPEAKER	< Speaker

	A2DP_SOURCE_FEATURE_MICROPHONE	< Mic
	A2DP_SOURCE_FEATURE_MIXER	< Mixer
	A2DP_SOURCE_FEATURE_PLAYER	< Player
	A2DP_SOURCE_FEATURE_TUNER	< Tuner
	AAC_CHANNELS_1	This is macro AAC_CHANNELS_1.
	AAC_CHANNELS_2	This is macro AAC_CHANNELS_2.
	AAC_CHANNELS_ALL	This is macro AAC_CHANNELS_ALL.
	AAC_OBJECT_TYPE_MPEG_2_LC	This is macro AAC_OBJECT_TYPE_MPEG_2_LC.
	AAC_OBJECT_TYPE_MPEG_4_LC	This is macro AAC_OBJECT_TYPE_MPEG_4_LC.
	AAC_OBJECT_TYPE_MPEG_4_LTP	This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.
	AAC_OBJECT_TYPE_MPEG_4_SCALABLE	This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.
	AAC_SAMPLING_FREQUENCY_11025	This is macro AAC_SAMPLING_FREQUENCY_11025.
	AAC_SAMPLING_FREQUENCY_12000	This is macro AAC_SAMPLING_FREQUENCY_12000.
	AAC_SAMPLING_FREQUENCY_16000	This is macro AAC_SAMPLING_FREQUENCY_16000.
	AAC_SAMPLING_FREQUENCY_22050	This is macro AAC_SAMPLING_FREQUENCY_22050.
	AAC_SAMPLING_FREQUENCY_24000	This is macro AAC_SAMPLING_FREQUENCY_24000.
	AAC_SAMPLING_FREQUENCY_32000	This is macro AAC_SAMPLING_FREQUENCY_32000.
	AAC_SAMPLING_FREQUENCY_44100	This is macro AAC_SAMPLING_FREQUENCY_44100.
	AAC_SAMPLING_FREQUENCY_48000	This is macro AAC_SAMPLING_FREQUENCY_48000.
	AAC_SAMPLING_FREQUENCY_64000	This is macro AAC_SAMPLING_FREQUENCY_64000.
	AAC_SAMPLING_FREQUENCY_8000	This is macro AAC_SAMPLING_FREQUENCY_8000.
	AAC_SAMPLING_FREQUENCY_88200	This is macro AAC_SAMPLING_FREQUENCY_88200.
	AAC_SAMPLING_FREQUENCY_96000	This is macro AAC_SAMPLING_FREQUENCY_96000.
	AAC_SAMPLING_FREQUENCY_ALL	This is macro AAC_SAMPLING_FREQUENCY_ALL.
	AAC_VBR_NOT_SUPPORTED	This is macro AAC_VBR_NOT_SUPPORTED.
	AAC_VBR_SUPPORTED	This is macro AAC_VBR_SUPPORTED.
	bt_a2dp_abort_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_a2dp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up... more
	bt_a2dp_call_codec	This is macro bt_a2dp_call_codec.

	bt_a2dp_cancel_listen	brief Cancel listening for incoming connections. ingroup a2dp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_close_stream	brief Close a stream. ingroup a2dp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_a2dp_connect	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more
	bt_a2dp_connect_ex	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more
	bt_a2dp_create_stream	brief Create a stream. ingroup a2dp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr A2DP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_a2dp_destroy_stream	brief Destroy a stream. ingroup a2dp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_a2dp_disconnect	brief Disconnect from a remote device. ingroup a2dp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_a2dp_discover	brief Discover SEPs on a remote device. ingroup a2dp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr A2DP... more
	bt_a2dp_find_codec	brief Find a codec ingroup a2dp details A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in... more
	bt_a2dp_get_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_hci_connection	brief Get HCI connection for a stream ingroup a2dp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param mgr A2DP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p... more

	bt_a2dp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup a2dp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr A2DP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more
	bt_a2dp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup a2dp details This function returns the type of the codec currently used with the stream. param mgr A2DP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_a2dp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup a2dp details This function returns the ID of the local SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_remote_address	brief Get stream's remote BT address. ingroup a2dp details This function returns the address of the remote device associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_a2dp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup a2dp details This function returns the ID of the remote SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_state	brief Get local stream state. ingroup a2dp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr A2DP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more

	bt_a2dp_listen	brief Listen for incoming connections. ingroup a2dp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more
	bt_a2dp_reconfigure_stream	brief Reconfigure stream. ingroup a2dp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_a2dp_register_sink	brief Register a Sink SEP with the local A2DP manager. ingroup a2dp details This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_register_source	brief Register a Source SEP with the local A2DP manager. ingroup a2dp details This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling ::bt_a2dp_remove_media_tx_buffer. param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more

	bt_a2dp_start_stream	brief Start a stream. ingroup a2dp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more
	bt_a2dp_suspend_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more
	MPEG_BITRATE_0000	This is macro MPEG_BITRATE_0000.
	MPEG_BITRATE_0001	This is macro MPEG_BITRATE_0001.
	MPEG_BITRATE_0010	This is macro MPEG_BITRATE_0010.
	MPEG_BITRATE_0011	This is macro MPEG_BITRATE_0011.
	MPEG_BITRATE_0100	This is macro MPEG_BITRATE_0100.
	MPEG_BITRATE_0101	This is macro MPEG_BITRATE_0101.
	MPEG_BITRATE_0110	This is macro MPEG_BITRATE_0110.
	MPEG_BITRATE_0111	This is macro MPEG_BITRATE_0111.
	MPEG_BITRATE_1000	This is macro MPEG_BITRATE_1000.
	MPEG_BITRATE_1001	This is macro MPEG_BITRATE_1001.
	MPEG_BITRATE_1010	This is macro MPEG_BITRATE_1010.
	MPEG_BITRATE_1011	This is macro MPEG_BITRATE_1011.
	MPEG_BITRATE_1100	This is macro MPEG_BITRATE_1100.
	MPEG_BITRATE_1101	This is macro MPEG_BITRATE_1101.
	MPEG_BITRATE_1110	This is macro MPEG_BITRATE_1110.
	MPEG_BITRATE_ALL	This is macro MPEG_BITRATE_ALL.
	MPEG_CHANNEL_MODE_ALL	This is macro MPEG_CHANNEL_MODE_ALL.
	MPEG_CHANNEL_MODE_DUAL_CHANNEL	This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.
	MPEG_CHANNEL_MODE_JOINT_STEREO	This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.
	MPEG_CHANNEL_MODE_MONO	This is macro MPEG_CHANNEL_MODE_MONO.
	MPEG_CHANNEL_MODE_STEREO	This is macro MPEG_CHANNEL_MODE_STEREO.
	MPEG_CRC_PROTECTION_NOT_SUPPORTED	This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.
	MPEG_CRC_PROTECTION_SUPPORTED	This is macro MPEG_CRC_PROTECTION_SUPPORTED.
	MPEG_LAYER_1	This is macro MPEG_LAYER_1.
	MPEG_LAYER_2	This is macro MPEG_LAYER_2.
	MPEG_LAYER_3	This is macro MPEG_LAYER_3.
	MPEG_LAYER_ALL	This is macro MPEG_LAYER_ALL.
	MPEG_MPF_1	This is macro MPEG_MPF_1.
	MPEG_MPF_2	This is macro MPEG_MPF_2.
	MPEG_SAMPLING_FREQUENCY_16000	This is macro MPEG_SAMPLING_FREQUENCY_16000.
	MPEG_SAMPLING_FREQUENCY_22050	This is macro MPEG_SAMPLING_FREQUENCY_22050.
	MPEG_SAMPLING_FREQUENCY_24000	This is macro MPEG_SAMPLING_FREQUENCY_24000.
	MPEG_SAMPLING_FREQUENCY_32000	This is macro MPEG_SAMPLING_FREQUENCY_32000.
	MPEG_SAMPLING_FREQUENCY_44100	This is macro MPEG_SAMPLING_FREQUENCY_44100.

	MPEG_SAMPLING_FREQUENCY_48000	This is macro MPEG_SAMPLING_FREQUENCY_48000.
	MPEG_SAMPLING_FREQUENCY_ALL	This is macro MPEG_SAMPLING_FREQUENCY_ALL.
	MPEG_VBR_NOT_SUPPORTED	This is macro MPEG_VBR_NOT_SUPPORTED.
	MPEG_VBR_SUPPORTED	This is macro MPEG_VBR_SUPPORTED.
	SBC_ALLOCATION_METHOD_ALL	This is macro SBC_ALLOCATION_METHOD_ALL.
	SBC_ALLOCATION_METHOD_LOUDNESS	This is macro SBC_ALLOCATION_METHOD_LOUDNESS.
	SBC_ALLOCATION_METHOD_SNR	This is macro SBC_ALLOCATION_METHOD_SNR.
	SBC_BLOCK_LENGTH_12	This is macro SBC_BLOCK_LENGTH_12.
	SBC_BLOCK_LENGTH_16	This is macro SBC_BLOCK_LENGTH_16.
	SBC_BLOCK_LENGTH_4	This is macro SBC_BLOCK_LENGTH_4.
	SBC_BLOCK_LENGTH_8	This is macro SBC_BLOCK_LENGTH_8.
	SBC_BLOCK_LENGTH_ALL	This is macro SBC_BLOCK_LENGTH_ALL.
	SBC_CHANNEL_MODE_ALL	This is macro SBC_CHANNEL_MODE_ALL.
	SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.
	SBC_CHANNEL_MODE_JOINT_STEREO	This is macro SBC_CHANNEL_MODE_JOINT_STEREO.
	SBC_CHANNEL_MODE_MONO	This is macro SBC_CHANNEL_MODE_MONO.
	SBC_CHANNEL_MODE_STEREO	This is macro SBC_CHANNEL_MODE_STEREO.
	SBC_SAMPLING_FREQUENCY_16000	This is macro SBC_SAMPLING_FREQUENCY_16000.
	SBC_SAMPLING_FREQUENCY_32000	This is macro SBC_SAMPLING_FREQUENCY_32000.
	SBC_SAMPLING_FREQUENCY_44100	This is macro SBC_SAMPLING_FREQUENCY_44100.
	SBC_SAMPLING_FREQUENCY_48000	This is macro SBC_SAMPLING_FREQUENCY_48000.
	SBC_SAMPLING_FREQUENCY_ALL	This is macro SBC_SAMPLING_FREQUENCY_ALL.
	SBC_SUBBANDS_4	This is macro SBC_SUBBANDS_4.
	SBC_SUBBANDS_8	This is macro SBC_SUBBANDS_8.
	SBC_SUBBANDS_ALL	This is macro SBC_SUBBANDS_ALL.
	bt_a2dp_get_all_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_configuration	This is macro bt_a2dp_get_configuration.
	bt_a2dp_open_stream	This is macro bt_a2dp_open_stream.
	A2DP_EVT_SET_STREAM_CONFIGURATION	This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.
	bt_a2dp_clear_media_tx_queue	brief Clear send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_a2dp_clear_media_tx_queue. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more

	bt_a2dp_get_stream_config	brief Get stream's configuration. ingroup a2dp details This function returns a pointer to a structure holding current configuration of the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_a2dp_get_stream_direction	This is macro bt_a2dp_get_stream_direction.
	bt_a2dp_report_delay	This is macro bt_a2dp_report_delay.
	bt_a2dp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more

A2DP Functions

	Name	Description
	_bt_a2dp_avdtp_mgr_callback	This is function _bt_a2dp_avdtp_mgr_callback.
	bt_a2dp_aac_codec_handler	This is function bt_a2dp_aac_codec_handler.
	bt_a2dp_find_server	This is function bt_a2dp_find_server.
	bt_a2dp_find_sink	brief Find sink ingroup a2dp details This function looks for a sink on a remote device specified by c deviceAddress and, if found, returns features supported by the sink. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_find_source	brief Find source ingroup a2dp details This function looks for a source on a remote device specified by c deviceAddress and, if found, returns features supported by the source. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_get_mgr	brief Return a pointer to an instance of the A2DP manager. ingroup a2dp details This function returns a pointer to an instance of the A@DP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.
	bt_a2dp_init	brief Initialize the A2DP layer. ingroup a2dp details This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.
	bt_a2dp_mpeg_codec_handler	This is function bt_a2dp_mpeg_codec_handler.
	bt_a2dp_open_and_start_stream	brief Open & start a stream ingroup a2dp details Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with... more

	bt_a2dp_register_aac_codec	brief Register default AAC codec ingroup a2dp details This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_register_callback	brief Register a A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.... more
	bt_a2dp_register_mpeg_codec	brief Register default MPEG codec ingroup a2dp details This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_sbc_codec_handler	This is function bt_a2dp_sbc_codec_handler.
	bt_a2dp_start	brief Start the A2DP layer. ingroup a2dp details This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_set_configuration	This is function bt_a2dp_set_configuration.

AVCTP Data Types and Constants

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

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

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

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

AVCTP Functions

	Name	Description
	_bt_avctp_allocate_channel	This is function _bt_avctp_allocate_channel.
	_bt_avctp_allocate_message	This is function _bt_avctp_allocate_message.
	_bt_avctp_allocate_transport	This is function _bt_avctp_allocate_transport.
	_bt_avctp_find_channel	This is function _bt_avctp_find_channel.
	_bt_avctp_find_transport	This is function _bt_avctp_find_transport.
	_bt_avctp_free_channel	This is function _bt_avctp_free_channel.
	_bt_avctp_free_message	This is function _bt_avctp_free_message.
	_bt_avctp_free_transport	This is function _bt_avctp_free_transport.
	_bt_avctp_init_message_buffers	This is function _bt_avctp_init_message_buffers.
	_bt_avctp_init_signal	This is function _bt_avctp_init_signal.
	_bt_avctp_l2cap_read_data_callback	This is function _bt_avctp_l2cap_read_data_callback.
	_bt_avctp_packet_assembler	This is function _bt_avctp_packet_assembler.

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

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

AVDTP Data Types and Constants

	Name	Description
	_bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_param_u	brief Parameter to a codec handler. ingroup avdtp details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.

	_bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	_bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	_bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	_bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	_bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	_bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	_bt_avdtp_event_u	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	_bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	_bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	_bt_avdtp_evt_ctrl_channel_connected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.

	_bt_avdtp_evt_ctrl_channel_disconnected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	_bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	_bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	_bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	_bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	_bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	_bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.

	_bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	_bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	_bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	_bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	_bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	_bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

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

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

	bt_avdtp_codec_op_param_t	This is type bt_avdtp_codec_op_param_t.
	bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	bt_avdtp_event_t	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	bt_avdtp_evt_ctrl_channel_connected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.

	bt_avdtp_evt_ctrl_channel_disconnected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.

	bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

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

	bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.
	bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	bt_avdtp_mgr_callback_fp	brief AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr AVDTP manager. param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED:... more
	bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	bt_avdtp_sep_t	This is type bt_avdtp_sep_t.
	bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	bt_avdtp_transport_op_callback_fp	This is type bt_avdtp_transport_op_callback_fp.
	bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.
	bt_media_packet_t	This is type bt_media_packet_t.
	__AVDTP_CONFIG_H	This is macro __AVDTP_CONFIG_H.

	__AVDTP_CONTROL_H	This is macro __AVDTP_CONTROL_H.
	__AVDTP_H	This is macro __AVDTP_H.
	__AVDTP_PRIVATE_H	This is macro __AVDTP_PRIVATE_H.
	AVDTP_CODEC_OPCODE_PARSE_CONFIG	< Parse codec configuration.
	AVDTP_CODEC_OPCODE_PARSE_PACKET	This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.
	AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG	< Serialize codec configuration.
	AVDTP_CODEC_TYPE_ATRAC	< ATRAC (proprietary codec owned by Sony Corporation).
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO	< MPEG-1,2 (optional).
	AVDTP_CODEC_TYPE_MPEG2_4_AAC	< MPEG-2,4 AAC (optional, used in Apple's products).
	AVDTP_CODEC_TYPE_NON_A2DP	< Vendor specific.
	AVDTP_CODEC_TYPE_SBC	< SBC (mandatory to support in A2DP profile).
	AVDTP_CTLR_CHANNEL_EVT_CONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.
	AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_NOTHING	This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.
	AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED	This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTING	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.
	AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.
	AVDTP_CTRL_MESSAGE_TYPE_ACCEPT	This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.
	AVDTP_CTRL_MESSAGE_TYPE_COMMAND	This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.
	AVDTP_CTRL_MESSAGE_TYPE_FLD	This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.
	AVDTP_CTRL_MESSAGE_TYPE_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.
	AVDTP_CTRL_PACKET_TYPE_CONTINUE	This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.
	AVDTP_CTRL_PACKET_TYPE_END	This is macro AVDTP_CTRL_PACKET_TYPE_END.
	AVDTP_CTRL_PACKET_TYPE_FLD	This is macro AVDTP_CTRL_PACKET_TYPE_FLD.
	AVDTP_CTRL_PACKET_TYPE_SIGNLE	This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.
	AVDTP_CTRL_PACKET_TYPE_START	This is macro AVDTP_CTRL_PACKET_TYPE_START.
	AVDTP_ERROR_BAD_ACP_SEID	< The requested command indicates an invalid ACP SEP ID (not addressable)
	AVDTP_ERROR_BAD_CP_FORMAT	< The format of Content Protection Service Capability is not correct.
	AVDTP_ERROR_BAD_HEADER_FORMAT	< The request packet header format is invalid.
	AVDTP_ERROR_BAD_LENGTH	< The request packet length is not match the assumed length.
	AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT	< The format of Media Transport Capability is not correct.
	AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT	< The format of Multiplexing Service Capability is not correct.
	AVDTP_ERROR_BAD_PAYLOAD_FORMAT	< The requested command has an incorrect payload format.
	AVDTP_ERROR_BAD_RECOVERY_FORMAT	< The format of Recovery Service Capability is not correct.
	AVDTP_ERROR_BAD_RECOVERY_TYPE	< The requested Recovery Type is not defined in AVDTP.
	AVDTP_ERROR_BAD_ROHC_FORMAT	< The format of Header Compression Service Capability is not correct.
	AVDTP_ERROR_BAD_SERV_CATEGORY	< The value of Service Category in the request packet is not defined in AVDTP.
	AVDTP_ERROR_BAD_STATE	< The stream is in state that does not permit executing commands.
	AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL	< An attempt to establish a control channel has failed.
	AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT	< An attempt to establish a transport channel has failed.
	AVDTP_ERROR_INVALID_CAPABILITIES	< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.
	AVDTP_ERROR_NOT_SUPPORTED_COMMAND	< The requested command is not supported by the device.
	AVDTP_ERROR_SEP_IN_USE	< The SEP is in use.

	AVDTP_ERROR_SEP_NOT_IN_USE	< The SEP is not in use.
	AVDTP_ERROR_SUCCESS	< The operation completed with no errors.
	AVDTP_ERROR_UNSUPPORTED_CONFIGURAION	< Configuration not supported.
	AVDTP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	AVDTP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	AVDTP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	AVDTP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	AVDTP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	AVDTP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	AVDTP_EVT_DISCOVER_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	AVDTP_EVT_LAST	This is macro AVDTP_EVT_LAST.
	AVDTP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	AVDTP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	AVDTP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	AVDTP_EVT_NULL	This is macro AVDTP_EVT_NULL.
	AVDTP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	AVDTP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	AVDTP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	AVDTP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	AVDTP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	AVDTP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	AVDTP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.

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

	AVDTP_SEP_CAPABILITY_FLAG_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_STATE_FREE	This is macro AVDTP_SEP_STATE_FREE.
	AVDTP_SEP_STATE_IDLE	This is macro AVDTP_SEP_STATE_IDLE.
	AVDTP_SEP_TYPE_SINK	< Sink (usually a device like a headphones or BMW).
	AVDTP_SEP_TYPE_SOURCE	< Source (usually a device like a phone, desktop or laptop).
	AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS	< The stream is closing transport channels.
	AVDTP_STREAM_FLAG_LISTENING	This is macro AVDTP_STREAM_FLAG_LISTENING.
	AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS	< The stream is opening transport channels.
	AVDTP_STREAM_STATE_ABORTING	< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CLOSING	< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CONFIGURED	< The stream has been configured.
	AVDTP_STREAM_STATE_IDLE	< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.
	AVDTP_STREAM_STATE_OPEN	< The stream has been opened.
	AVDTP_STREAM_STATE_STREAMING	< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.
	AVDTP_TC_OPCODE_CONNECT	This is macro AVDTP_TC_OPCODE_CONNECT.
	AVDTP_TC_OPCODE_DISCONNECT	This is macro AVDTP_TC_OPCODE_DISCONNECT.
	AVDTP_TC_STATUS_ERROR	This is macro AVDTP_TC_STATUS_ERROR.
	AVDTP_TC_STATUS_SUCCESS	This is macro AVDTP_TC_STATUS_SUCCESS.
	AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.
	AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.
	AVDTP_TRANSPORT_SESSION_TYPE_MEDIA	< Media (audio or video).
	AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY	< Recovery (currently not supported).
	AVDTP_TRANSPORT_SESSION_TYPE_REPORTING	< Reporting (currently not supported).

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

	_bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_ctrl_connection_failed_t	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	AVDTP_EVT_DELAYREPORT_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION	This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.
	AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING	< Delat reporitng.
	AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING	< Delay Reporting.

AVDTP Functions

	Name	Description
	_bt_avdtp_add_param_byte	This is function _bt_avdtp_add_param_byte.
	_bt_avdtp_add_param_uint	This is function _bt_avdtp_add_param_uint.
	_bt_avdtp_allocate_buffers	This is function _bt_avdtp_allocate_buffers.
	_bt_avdtp_allocate_cmd	This is function _bt_avdtp_allocate_cmd.
	_bt_avdtp_allocate_sep	This is function _bt_avdtp_allocate_sep.
	_bt_avdtp_allocate_sep_config	This is function _bt_avdtp_allocate_sep_config.
	_bt_avdtp_allocate_stream	This is function _bt_avdtp_allocate_stream.
	_bt_avdtp_allocate_transport_channel	This is function _bt_avdtp_allocate_transport_channel.
	_bt_avdtp_allocate_transport_session_id	This is function _bt_avdtp_allocate_transport_session_id.
	_bt_avdtp_begin_tc_channel_operation	This is function _bt_avdtp_begin_tc_channel_operation.
	_bt_avdtp_commit_tc_channel_operation	This is function _bt_avdtp_commit_tc_channel_operation.
	_bt_avdtp_control_channel_accept_handler	This is function _bt_avdtp_control_channel_accept_handler.
	_bt_avdtp_control_channel_cmd_handler	This is function _bt_avdtp_control_channel_cmd_handler.
	_bt_avdtp_control_channel_event_handler	This is function _bt_avdtp_control_channel_event_handler.
	_bt_avdtp_control_channel_reject_handler	This is function _bt_avdtp_control_channel_reject_handler.
	_bt_avdtp_execute_tc_channel_operation	This is function _bt_avdtp_execute_tc_channel_operation.
	_bt_avdtp_find_listening_stream	This is function _bt_avdtp_find_listening_stream.
	_bt_avdtp_find_stream	This is function _bt_avdtp_find_stream.
	_bt_avdtp_find_stream_by_remote_sep_id	This is function _bt_avdtp_find_stream_by_remote_sep_id.
	_bt_avdtp_find_stream_by_sep_id	This is function _bt_avdtp_find_stream_by_sep_id.
	_bt_avdtp_free_cmd	This is function _bt_avdtp_free_cmd.
	_bt_avdtp_free_sep_config	This is function _bt_avdtp_free_sep_config.
	_bt_avdtp_free_stream	This is function _bt_avdtp_free_stream.
	_bt_avdtp_free_transport_channel	This is function _bt_avdtp_free_transport_channel.
	_bt_avdtp_get_control_channel	This is function _bt_avdtp_get_control_channel.
	_bt_avdtp_init_cmd_buffers	This is function _bt_avdtp_init_cmd_buffers.
	_bt_avdtp_init_sep_config_buffers	This is function _bt_avdtp_init_sep_config_buffers.

	_bt_avdtp_init_signal	This is function _bt_avdtp_init_signal.
	_bt_avdtp_is_sep_inuse	This is function _bt_avdtp_is_sep_inuse.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_register_transport_channel_for_operation	This is function _bt_avdtp_register_transport_channel_for_operation.
	_bt_avdtp_send_command	This is function _bt_avdtp_send_command.
	_bt_avdtp_send_media_packet	This is function _bt_avdtp_send_media_packet.
	_bt_avdtp_set_signal	This is function _bt_avdtp_set_signal.
	_bt_avdtp_transport_l2cap_read_data_callback	This is function _bt_avdtp_transport_l2cap_read_data_callback.
	_bt_avdtp_transport_l2cap_state_changed_callback	This is function _bt_avdtp_transport_l2cap_state_changed_callback.
	_bt_avdtp_write_caps	This is function _bt_avdtp_write_caps.
	bt_avdtp_abort_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_avdtp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_avdtp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a AVDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up... more
	bt_avdtp_cancel_listen	brief Cancel listening for incoming connections. ingroup avdtp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_close_stream	brief Close a stream. ingroup avdtp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_avdtp_create_stream	brief Create a stream. ingroup avdtp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr AVDTP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.

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

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

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

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

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

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

AVRCP Data Types and Constants

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

AVRCP Functions

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

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

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

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

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

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

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

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

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

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

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

Bluetooth Support Data Types and Constants

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

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

	bt_uuid_t	This is type bt_uuid_t.
	bt_uuid16	This is type bt_uuid16.
	bt_uuid32	This is type bt_uuid32.
	__BLUETOOTH_STG_H	This is macro __BLUETOOTH_STG_H.

Bluetooth Support Functions

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

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

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

GAP Data Types and Constants

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

GAP Functions

	Name	Description
	bt_gap_find_devices	This is function bt_gap_find_devices.

HCI Data Types and Constants

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

HCI Functions

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

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

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

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

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

	_hci_init_cmd_buffers	This is function _hci_init_cmd_buffers.
	_hci_init_cmd_queues	This is function _hci_init_cmd_queues.
	_hci_receive_start	From hci_receive.c
	_hci_recv_sco_data_packet	From hci_sco.c
	_hci_send_commands_from_queue	From hci_send.c
	_hci_send_data	This is function _hci_send_data.
	_hci_send_data_fragment	This is function _hci_send_data_fragment.
	_hci_send_data_from_queue	This is function _hci_send_data_from_queue.
	_bt_le_evt_handler	This is function _bt_le_evt_handler.
	bt_hci_le_allocate_set_scan_response_data_command	This is function bt_hci_le_allocate_set_scan_response_data_command.
	bt_hci_le_cancel_connect_ex	This is function bt_hci_le_cancel_connect_ex.
	bt_hci_le_ibeacon_add	This is function bt_hci_le_ibeacon_add.
	bt_hci_set_vendor_specific_event_handler	This is function bt_hci_set_vendor_specific_event_handler.
	bt_hcitr_3wire_init_ex	This is function bt_hcitr_3wire_init_ex.
	bt_hcitr_3wire_reset_ex	This is function bt_hcitr_3wire_reset_ex.
	_bt_hci_set_init_flags	This is function _bt_hci_set_init_flags.
	hci_get_conn_state_by_bdaddr	This is function hci_get_conn_state_by_bdaddr.
	bt_hci_cancel_find_devices_ex	brief Stop inquiry
	bt_hci_get_inquiry_response_tx_power_level	This is function bt_hci_get_inquiry_response_tx_power_level.
	bt_hci_init_ex	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.
	bt_hci_reset	brief Reset controller. ingroup hci details This function resets the BT controller. param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_send_pin_code_ex	This is function bt_hci_send_pin_code_ex.
	bt_hci_set_scan_ex	This is function bt_hci_set_scan_ex.
	bt_hci_write_link_policy_settings	This is function bt_hci_write_link_policy_settings.
	bt_hci_write_local_name_ex	This is function bt_hci_write_local_name_ex.
	bt_hcitr_3wire_cancel_recv_packet	This is function bt_hcitr_3wire_cancel_recv_packet.

L2CAP Data Types and Constants

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

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

	L2CAP_CMD_LAST	This is macro L2CAP_CMD_LAST.
	L2CAP_CMD_REJECT	This is macro L2CAP_CMD_REJECT.
	L2CAP_CMD_RESERVED	This is macro L2CAP_CMD_RESERVED.
	L2CAP_CMD_STATUS_BEING_SENT	This is macro L2CAP_CMD_STATUS_BEING_SENT.
	L2CAP_CMD_STATUS_PENDING	This is macro L2CAP_CMD_STATUS_PENDING.
	L2CAP_CMD_STATUS_WAITING_RESPONSE	This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.
	L2CAP_CONFIG_RESULT_REJECTED	This is macro L2CAP_CONFIG_RESULT_REJECTED.
	L2CAP_CONFIG_RESULT_SUCCESS	This is macro L2CAP_CONFIG_RESULT_SUCCESS.
	L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER	This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.
	L2CAP_CONFIG_RESULT_UNKNOWN_OPTION	This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.
	L2CAP_CONN_REQ_RESULT_INVALID_PSM	This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.
	L2CAP_CONN_REQ_RESULT_NO_RESOURCES	This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.
	L2CAP_CONN_REQ_RESULT_PENDING	This is macro L2CAP_CONN_REQ_RESULT_PENDING.
	L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK	This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.
	L2CAP_CONN_REQ_RESULT_SUCCESS	This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.
	L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.
	L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.
	L2CAP_CONN_REQ_STATUS_NO_INFO	This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.
	L2CAP_DEFAULT_ERTX	seconds
	L2CAP_DEFAULT_RTX	seconds
	L2CAP_ECHO_MAX_DATA_LEN	echo request and response
	L2CAP_ERETR_RECV_STATE_RECV	This is macro L2CAP_ERETR_RECV_STATE_RECV.
	L2CAP_ERETR_RECV_STATE_REJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.
	L2CAP_ERETR_RECV_STATE_SREJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.
	L2CAP_ERETR_XMIT_STATE_WAIT_ACK	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.
	L2CAP_ERETR_XMIT_STATE_WAIT_F	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.
	L2CAP_ERETR_XMIT_STATE_XMIT	This is macro L2CAP_ERETR_XMIT_STATE_XMIT.
	L2CAP_EXT_BI_QOS	This is macro L2CAP_EXT_BI_QOS.
	L2CAP_EXT_ENHANCED_RETRANSMISSION	This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.
	L2CAP_EXT_FCS_OPTION	This is macro L2CAP_EXT_FCS_OPTION.
	L2CAP_EXT_FLOW_CONTROL	This is macro L2CAP_EXT_FLOW_CONTROL.
	L2CAP_EXT_RETRANSMISSION	This is macro L2CAP_EXT_RETRANSMISSION.
	L2CAP_EXT_STREAMING	This is macro L2CAP_EXT_STREAMING.
	L2CAP_FRAME_TYPE_I	This is macro L2CAP_FRAME_TYPE_I.
	L2CAP_FRAME_TYPE_S	This is macro L2CAP_FRAME_TYPE_S.
	L2CAP_HCI_CONNECT_PACKET_TYPE	This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.
	L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE	This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.
	L2CAP_HCI_CONNECT_ROLE_SWITCH	This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.
	L2CAP_HEADER_LEN	This is macro L2CAP_HEADER_LEN.
	L2CAP_IDLE_HCI_CONNECTION_TIMEOUT	This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.
	L2CAP_INFO_NOT_SUPPORTED	REDFLAG: should it be non zero?
	L2CAP_INFO_RESULT_SUCCESS	This is macro L2CAP_INFO_RESULT_SUCCESS.
	L2CAP_INFO_TYPE_CONNECTIONLESS_MTU	information request
	L2CAP_INFO_TYPE_EXTENDED_SUPPORT	This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.

	L2CAP_LINK_TYPE_BD_EDR	This is macro L2CAP_LINK_TYPE_BD_EDR.
	L2CAP_LINK_TYPE_LE	This is macro L2CAP_LINK_TYPE_LE.
	L2CAP_MAX_CMD_QUEUE_LEN	This is macro L2CAP_MAX_CMD_QUEUE_LEN.
	L2CAP_MAX_FRAME_BUFFERS	This is macro L2CAP_MAX_FRAME_BUFFERS.
	L2CAP_MAX_MANAGERS	This is macro L2CAP_MAX_MANAGERS.
	L2CAP_MAX_MTU	This is macro L2CAP_MAX_MTU.
	L2CAP_MAX_OPTIONS	configuration request/response
	L2CAP_MAX_RTX	This is macro L2CAP_MAX_RTX.
	L2CAP_MGR_STATE_FREE	This is macro L2CAP_MGR_STATE_FREE.
	L2CAP_MGR_STATE_LISTENING_L2CAP	This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.
	L2CAP_MGR_STATE_USED	This is macro L2CAP_MGR_STATE_USED.
	L2CAP_MIN_MTU	This is macro L2CAP_MIN_MTU.
	L2CAP_MONITOR_TIMEOUT	This is macro L2CAP_MONITOR_TIMEOUT.
	L2CAP_OPTION_LEN_FLASH_QOS	This is macro L2CAP_OPTION_LEN_FLASH_QOS.
	L2CAP_OPTION_LEN_FLASH_RFC	This is macro L2CAP_OPTION_LEN_FLASH_RFC.
	L2CAP_OPTION_LEN_FLASH_TIMEOUT	This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.
	L2CAP_OPTION_LEN_MAX_MTU	This is macro L2CAP_OPTION_LEN_MAX_MTU.
	L2CAP_OPTION_LEN_UNKNOWN	This is macro L2CAP_OPTION_LEN_UNKNOWN.
	L2CAP_OPTION_LEN_UNKNOWN_DATA	21
	L2CAP_OPTION_TYPE_FLASH_QOS	This is macro L2CAP_OPTION_TYPE_FLASH_QOS.
	L2CAP_OPTION_TYPE_FLASH_QOS_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.
	L2CAP_OPTION_TYPE_FLASH_RFC	This is macro L2CAP_OPTION_TYPE_FLASH_RFC.
	L2CAP_OPTION_TYPE_FLASH_RFC_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.
	L2CAP_OPTION_TYPE_MAX_MTU	This is macro L2CAP_OPTION_TYPE_MAX_MTU.
	L2CAP_OPTION_TYPE_MAX_MTU_FLAG	This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.
	L2CAP_OPTION_TYPE_UNKNOWN_FLAG	This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.
	L2CAP_PACKET_DATA_TYPE_RAW	This is macro L2CAP_PACKET_DATA_TYPE_RAW.
	L2CAP_PACKET_DATA_TYPE_SMART	This is macro L2CAP_PACKET_DATA_TYPE_SMART.
	L2CAP_REJECT_REASON_INVALID_CHANNEL	This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.
	L2CAP_REJECT_REASON_MTU_EXCEEDED	This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.
	L2CAP_REJECT_REASON_NOT_UNDERSTOOD	This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.
	L2CAP_RETR_TIMEOUT	This is macro L2CAP_RETR_TIMEOUT.
	L2CAP_RFC_BASIC	This is macro L2CAP_RFC_BASIC.
	L2CAP_RFC_ENHANCED_RETRANSMISSION	This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.
	L2CAP_RFC_ERETR_TIMEOUT	3 secs
	L2CAP_RFC_ERETR_TX_WINDOW	This is macro L2CAP_RFC_ERETR_TX_WINDOW.
	L2CAP_RFC_FLOW_CONTROL	This is macro L2CAP_RFC_FLOW_CONTROL.
	L2CAP_RFC_MONITOR_TIMEOUT	12 secs
	L2CAP_RFC_RETRANSMISSION	This is macro L2CAP_RFC_RETRANSMISSION.
	L2CAP_RFC_STREAMING	This is macro L2CAP_RFC_STREAMING.
	L2CAP_SAR_SDU_CONTINUE	This is macro L2CAP_SAR_SDU_CONTINUE.
	L2CAP_SAR_SDU_START	This is macro L2CAP_SAR_SDU_START.
	L2CAP_SAR_SDU_STOP	This is macro L2CAP_SAR_SDU_STOP.
	L2CAP_SAR_UNSEGMENTED	This is macro L2CAP_SAR_UNSEGMENTED.
	L2CAP_SERVICE_TYPE_BEST_EFFORT	This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.
	L2CAP_SERVICE_TYPE_GUARANTEED	This is macro L2CAP_SERVICE_TYPE_GUARANTEED.
	L2CAP_SERVICE_TYPE_NO_TRAFFIC	This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.
	L2CAP_SFUNCTION_REJ	This is macro L2CAP_SFUNCTION_REJ.
	L2CAP_SFUNCTION_RNR	This is macro L2CAP_SFUNCTION_RNR.

	L2CAP_SFUNCTION_RR	This is macro L2CAP_SFUNCTION_RR.
	L2CAP_SFUNCTION_SREJ	This is macro L2CAP_SFUNCTION_SREJ.
	PSM_ATT	This is macro PSM_ATT.
	PSM_AVCTP	This is macro PSM_AVCTP.
	PSM_AVCTP_Browsing	This is macro PSM_AVCTP_Browsing.
	PSM_AVDTP	This is macro PSM_AVDTP.
	PSM_BNEP	This is macro PSM_BNEP.
	PSM_HID_Control	This is macro PSM_HID_Control.
	PSM_HID_Interrupt	This is macro PSM_HID_Interrupt.
	PSM_RFCOMM	This is macro PSM_RFCOMM.
	PSM_SDP	This is macro PSM_SDP.
	SET_F_BIT	This is macro SET_F_BIT.
	SET_FRAME_TYPE	This is macro SET_FRAME_TYPE.
	SET_P_BIT	This is macro SET_P_BIT.
	SET_REQ_SEQ	This is macro SET_REQ_SEQ.
	SET_S_FUNCTION	This is macro SET_S_FUNCTION.
	SET_SAR	This is macro SET_SAR.
	SET_TX_SEQ	This is macro SET_TX_SEQ.
	_bt_l2cap_start_monitor_timer	This is macro _bt_l2cap_start_monitor_timer.
	_bt_l2cap_start_monitor_timer_if_not_running	This is macro _bt_l2cap_start_monitor_timer_if_not_running.
	_bt_l2cap_start_retr_timer	This is macro _bt_l2cap_start_retr_timer.
	_bt_l2cap_start_retr_timer_if_not_running	This is macro _bt_l2cap_start_retr_timer_if_not_running.
	_bt_l2cap_stop_monitor_timer	This is macro _bt_l2cap_stop_monitor_timer.
	_bt_l2cap_stop_retr_timer	This is macro _bt_l2cap_stop_retr_timer.
	_bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	_bt_l2cap_channel_t	This is record _bt_l2cap_channel_t.
	_bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	_bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	_bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	_bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	_bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	_bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	_bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	_bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	_bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	_bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	_bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	_bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	_bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	_bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	_bt_l2cap_command_t	This is type bt_l2cap_command_t.
	_bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	_bt_l2cap_fixed_channel_s	This is type bt_l2cap_fixed_channel_t.
	_bt_l2cap_frame_desc_s	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	_bt_l2cap_mgr_s	This is type bt_l2cap_mgr_t.
	_bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	_bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	_bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	_bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	_bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	_bt_l2cap_packet_t	This is type bt_l2cap_packet_t.

	_bt_l2cap_psm_s	This is type bt_l2cap_psm_t.
	_bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	bt_l2cap_cfg_option_p	This is type bt_l2cap_cfg_option_p.
	bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	bt_l2cap_channel_t	This is type bt_l2cap_channel_t.
	bt_l2cap_cmd_assembler_fp	This is type bt_l2cap_cmd_assembler_fp.
	bt_l2cap_cmd_config_req_p	This is type bt_l2cap_cmd_config_req_p.
	bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	bt_l2cap_cmd_config_res_p	This is type bt_l2cap_cmd_config_res_p.
	bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	bt_l2cap_cmd_connection_req_p	This is type bt_l2cap_cmd_connection_req_p.
	bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	bt_l2cap_cmd_connection_res_p	This is type bt_l2cap_cmd_connection_res_p.
	bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	bt_l2cap_cmd_disconnection_req_p	This is type bt_l2cap_cmd_disconnection_req_p.
	bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	bt_l2cap_cmd_echo_req_p	This is type bt_l2cap_cmd_echo_req_p.
	bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	bt_l2cap_cmd_echo_res_p	This is type bt_l2cap_cmd_echo_res_p.
	bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	bt_l2cap_cmd_header_p	This is type bt_l2cap_cmd_header_p.
	bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	bt_l2cap_cmd_info_req_p	This is type bt_l2cap_cmd_info_req_p.
	bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	bt_l2cap_cmd_info_res_p	This is type bt_l2cap_cmd_info_res_p.
	bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	bt_l2cap_cmd_parser_fp	This is type bt_l2cap_cmd_parser_fp.
	bt_l2cap_cmd_reject_p	This is type bt_l2cap_cmd_reject_p.
	bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	bt_l2cap_command_t	This is type bt_l2cap_command_t.
	bt_l2cap_connect_callback_fp	This is type bt_l2cap_connect_callback_fp.
	bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	bt_l2cap_fixed_channel_t	This is type bt_l2cap_fixed_channel_t.
	bt_l2cap_frame_desc_t	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	bt_l2cap_listen_callback_fp	This is type bt_l2cap_listen_callback_fp.
	bt_l2cap_mgr_p	This is type bt_l2cap_mgr_p.
	bt_l2cap_mgr_t	This is type bt_l2cap_mgr_t.
	bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	bt_l2cap_packet_t	This is type bt_l2cap_packet_t.
	bt_l2cap_psm_t	This is type bt_l2cap_psm_t.
	bt_l2cap_read_data_callback_fp	This is type bt_l2cap_read_data_callback_fp.
	bt_l2cap_request_handler_fp	This is type bt_l2cap_request_handler_fp.
	bt_l2cap_response_handler_fp	This is type bt_l2cap_response_handler_fp.
	bt_l2cap_send_data_callback_fp	This is type bt_l2cap_send_data_callback_fp.

	bt_l2cap_state_changed_callback_fp	This is type bt_l2cap_state_changed_callback_fp.
	bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	cmd_disconnection_res	This is type cmd_disconnection_res.
	pcmd_disconnection_res	This is type pcmd_disconnection_res.
	pf_l2cap_cmd_callback	This is type pf_l2cap_cmd_callback.
	L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT	This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.
	L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR	This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.
	L2CAP_EXT_EXT_WINDOW_SIZE	This is macro L2CAP_EXT_EXT_WINDOW_SIZE.
	L2CAP_EXT_FEATURES_ENABLED	This is macro L2CAP_EXT_FEATURES_ENABLED.
	L2CAP_EXT_FIXED_CHANNELS	This is macro L2CAP_EXT_FIXED_CHANNELS.
	L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT	This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.
	L2CAP_INFO_TYPE_FIXED_CHANNELS	This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.
	bt_l2cap_channel_ext_t	This is type bt_l2cap_channel_ext_t.
	_bt_l2cap_channel_ext_t	This is record _bt_l2cap_channel_ext_t.
	_bt_l2cap_connect_params_s	This is record _bt_l2cap_connect_params_s.
	bt_l2cap_connect_params_t	This is type bt_l2cap_connect_params_t.
	L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID	This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.
	L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED	This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.

L2CAP Functions

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

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

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

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

Misc. Data Types and Constants

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

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

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

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

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

Misc. Functions

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

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

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

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

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

Platform Data Types and Constants

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

RFCOMM Data Types and Constants

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

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

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

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

RFCOMM Functions

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

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

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

SBC Decoder Data Types and Constants

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

SBC Decoder Functions

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

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

SDP Data Types and Constants

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

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

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

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

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

	SDP_CLSID_HCR_PRINT	This is macro SDP_CLSID_HCR_PRINT.
	SDP_CLSID_HCR_SCAN	This is macro SDP_CLSID_HCR_SCAN.
	SDP_ATTRID_HCRP_1284ID	This is macro SDP_ATTRID_HCRP_1284ID.

SDP Functions

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

SPP Data Types and Constants

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

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

SPP Functions

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

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

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

	bt_spp_set_local_modem_status	• Set local device's TS 07.10 control signals. Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a command to the remote device this function... more
	bt_spp_set_port_options	This is function bt_spp_set_port_options.
	bt_spp_set_rts	• Set local device's RS-232 RTS signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more
	_bt_spp_find_port	This is function _bt_spp_find_port.
	_bt_spp_handle_rx	This is function _bt_spp_handle_rx.
	_bt_spp_handle_tx	This is function _bt_spp_handle_tx.
	_bt_spp_rfcomm_read_data_callback	This is function _bt_spp_rfcomm_read_data_callback.
	_bt_spp_rfcomm_send_data_callback	This is function _bt_spp_rfcomm_send_data_callback.
	_bt_spp_client_init	This is function _bt_spp_client_init.
	bt_spp_cancel_listen	• Stop listening for incoming connections. This function stops the port to accept incoming connections on a given server channel. @param port Serial port. @param channel The RFCOMM server channel to accept connections on. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_find_server_ex	This is function bt_spp_find_server_ex.

SSP Data Types and Constants

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

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

SSP Functions

	Name	Description
	bt_ssp_evt_handler	This is function bt_ssp_evt_handler.
	bt_ssp_init	This is function bt_ssp_init.
	bt_ssp_read_local_oob_data	This is function bt_ssp_read_local_oob_data.
	bt_ssp_send_keypress_notification	This is function bt_ssp_send_keypress_notification.
	bt_ssp_send_user_confirmation	This is function bt_ssp_send_user_confirmation.
	bt_ssp_send_user_passkey	This is function bt_ssp_send_user_passkey.
	bt_ssp_set_io_capabilities	This is function bt_ssp_set_io_capabilities.
	bt_ssp_set_mode	This is function bt_ssp_set_mode.
	bt_ssp_set_oob_data	This is function bt_ssp_set_oob_data.
	ssp_evt_io_capability_request	This is function ssp_evt_io_capability_request.
	ssp_evt_io_capability_response	This is function ssp_evt_io_capability_response.
	ssp_evt_keypress_notification	This is function ssp_evt_keypress_notification.
	ssp_evt_oob_data_request	This is function ssp_evt_oob_data_request.
	ssp_evt_ssp_complete	This is function ssp_evt_ssp_complete.
	ssp_evt_user_confirmation_request	This is function ssp_evt_user_confirmation_request.
	ssp_evt_user_passkey_notification	This is function ssp_evt_user_passkey_notification.
	ssp_evt_user_passkey_request	This is function ssp_evt_user_passkey_request.

Utility Data Types and Constants

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

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

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

	bt_xml_parser_callback_pf	This is type bt_xml_parser_callback_pf.
	bt_xml_parser_t	This is type bt_xml_parser_t.

Utility Functions

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

Description

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

Bluetooth Support Functions

bt_bdaddr_is_null Function

File

bt_bdaddr.h

C

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

Description

This is function bt_bdaddr_is_null.

bt_bdaddrs_are_equal Function

File

bt_bdaddr.h

C

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

Description

This is function bt_bdaddrs_are_equal.

bt_log_bdaddr Function

File

bt_log.h

C

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

Description

This is function bt_log_bdaddr.

bt_log_int Function

File

bt_log.h

C

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

Description

This is function bt_log_int.

bt_log_linkkey Function

File

bt_log.h

C

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

Description

This is function bt_log_linkkey.

bt_log_memory Function

File

bt_log.h

C

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

Description

This is function bt_log_memory.

bt_log_msg Function

File

bt_log.h

C

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

Description

This is function bt_log_msg.

bt_oem_assert Function

File

bt_oem.h

C

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

Description

This is function bt_oem_assert.

bt_oem_get_device_class Function

File

bt_oem.h

C

```
bt_long bt_oem_get_device_class();
```

Description

This is function bt_oem_get_device_class.

bt_oem_get_device_name Function

File

bt_oem.h

C

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

Description

This is function bt_oem_get_device_name.

bt_oem_get_pin_code Function

File

bt_oem.h

C

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

Description

This is function bt_oem_get_pin_code.

bt_oem_linkkey_notification Function

File

bt_oem.h

C

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

Description

This is function bt_oem_linkkey_notification.

bt_oem_linkkey_request Function

File

bt_oem.h

C

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

Description

This is function bt_oem_linkkey_request.

bt_oem_log_write Function

File

bt_log.h

C

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

Description

brief Output log message. ingroup log

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

bt_oem_recv Function

File

bt_hcitr.h

C

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

Description

- Receive data.

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

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

@param len Number of bytes to receive.

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

bt_oem_schedule_signals Function

File

bt_oem.h

C

```
void bt_oem_schedule_signals();
```

Description

This is function bt_oem_schedule_signals.

bt_oem_send Function

File

bt_hcitr.h

C

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

Description

- Send data.

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

@param buffer Pointer to the data to be sent .

@param len Number of bytes to send.

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

bt_oem_storage_get_capacity Function

File

bt_storage.h

C

```
bt_uint bt_oem_storage_get_capacity();
```

Description

brief Get non-volatile storage capacity. ingroup stg

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

bt_oem_storage_read Function

File

bt_storage.h

C

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

Description

brief Read from the non-volatile storage. ingroup stg

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

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

bt_oem_storage_start Function

File

bt_storage.h

C

```
void bt_oem_storage_start();
```

Description

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

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

bt_oem_storage_stop Function

File

bt_storage.h

C

```
void bt_oem_storage_stop();
```

Description

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

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

bt_oem_storage_write Function

File

bt_storage.h

C

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

Description

brief Write to non-volatile storage. ingroup stg

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

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

bt_oem_timer_clear Function

File

bt_timer.h

C

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

Description

brief Clear timer.

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

param timerId ID of the timer to clear.

bt_oem_timer_set Function

File

bt_timer.h

C

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

Description

brief Set timer.

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

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

bt_signal_process_pending Function

File

bt_signal.h

C

```
void bt_signal_process_pending();
```

Description

This is function bt_signal_process_pending.

bt_signal_register Function

File

bt_signal.h

C

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

Description

This is function bt_signal_register.

bt_signal_set Function

File

bt_signal.h

C

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

Description

This is function bt_signal_set.

bt_signal_unregister Function

File

bt_signal.h

C

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

Description

This is function bt_signal_unregister.

bt_sys_get_connectable Function

File

bt_system.h

C

```
bt_bool bt_sys_get_connectable();
```

Description

This is function bt_sys_get_connectable.

bt_sys_get_discoverable Function

File

bt_system.h

C

```
bt_bool bt_sys_get_discoverable();
```

Description

This is function bt_sys_get_discoverable.

bt_sys_get_l2cap_manager Function

File

bt_system.h

C

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

Description

- Get the L2CAP manager.

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

@return The L2CAP manager.

bt_sys_init Function

File

bt_system.h

C

```
void bt_sys_init();
```

Description

- Initialize the Bluetooth system.

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

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

bt_sys_set_modes Function

File

bt_system.h

C

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

```
callback_param);
```

Description

This is function bt_sys_set_modes.

bt_sys_start Function

File

bt_system.h

C

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

Description

- Start the Bluetooth system.

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

Also, the caller must provide an SDP database.

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

HCITR_BCSP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCITR_BCSP_ALLOCATE_BUFFERS();
```

Description

This is function HCITR_BCSP_ALLOCATE_BUFFERS.

IAPEA_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAPEA_ALLOCATE_BUFFERS();
```

Description

This is function IAPEA_ALLOCATE_BUFFERS.

bt_oem_ssp_callback Function

File

bt_oem.h

C

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

Description

This is function bt_oem_ssp_callback.

bt_signal_init Function

File

bt_signal.h

C

```
void bt_signal_init();
```

Description

This is function bt_signal_init.

bt_sys_init_ex Function

File

bt_system.h

C

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

Description

- Initialize the Bluetooth system.

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

Also, the caller must provide an SDP database.

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

To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

HCRP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCRP_ALLOCATE_BUFFERS();
```

Description

This is function HCRP_ALLOCATE_BUFFERS.

MAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
MAP_ALLOCATE_BUFFERS();
```

Description

This is function MAP_ALLOCATE_BUFFERS.

bt_q_contains Function

File

queue.h

C

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

Description

This is function bt_q_contains.

Bluetooth Support Data Types and Constants

ARG_NOT_USED Macro

File

bt_private.h

C

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

Description

This is macro ARG_NOT_USED.

BT_ASSERT Macro

File

bt_private.h

C

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

Description

This is macro BT_ASSERT.

BT_FALSE Macro

File

bt_types.h

C

```
#define BT_FALSE 0
```

Description

This is macro BT_FALSE.

BT_LINKKEY_LENGTH Macro

File

bt_types.h

C

```
#define BT_LINKKEY_LENGTH 16
```

Description

This is macro BT_LINKKEY_LENGTH.

BT_LOG Macro

File

bt_log.h

C

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

Description

This is macro BT_LOG.

BT_LOG_EX Macro

File

bt_log.h

C

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

Description

This is macro BT_LOG_EX.

BT_LOG_LEVEL_ALL Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_ALL 0
```

Description

This is macro BT_LOG_LEVEL_ALL.

BT_LOG_LEVEL_DEBUG Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_DEBUG 80
```

Description

This is macro BT_LOG_LEVEL_DEBUG.

BT_LOG_LEVEL_ERROR Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_ERROR 240
```

Description

This is macro BT_LOG_LEVEL_ERROR.

BT_LOG_LEVEL_INFO Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_INFO 160
```

Description

This is macro BT_LOG_LEVEL_INFO.

BT_LOG_LEVEL_OFF Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_OFF 0xFF
```

Description

This is macro BT_LOG_LEVEL_OFF.

BT_LOGADDR Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGADDR.

BT_LOGADDR_EX Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGADDR_EX.

BT_LOGINT Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGINT.

BT_LOGINT_EX Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGINT_EX.

BT_LOGLINKKEY Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGLINKKEY.

BT_LOGLINKKEY_EX Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGLINKKEY_EX.

BT_LOGMEMORY Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGMEMORY.

BT_LOGMEMORY_EX Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGMEMORY_EX.

BT_LOGWRITE Macro

File

bt_log.h

C

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

Description

This is macro BT_LOGWRITE.

BT_MAKE_BDADDR Macro

File

bt_types.h

C

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

Description

This is macro BT_MAKE_BDADDR.

BT_MAKE_BDADDR_LE Macro

File

bt_types.h

C

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

Description

This is macro BT_MAKE_BDADDR_LE.

BT_NO Macro

File

bt_types.h

C

```
#define BT_NO 0
```

Description

This is macro BT_NO.

BT_TRUE Macro

File

bt_types.h

C

```
#define BT_TRUE 1
```

Description

This is macro BT_TRUE.

BT_YES Macro

File

bt_types.h

C

```
#define BT_YES 1
```

Description

This is macro BT_YES.

CDS_LAST_DEVICE_ADDR Macro

File

bt_storage.h

C

```
#define CDS_LAST_DEVICE_ADDR HIDS_LAST_DEVICE_ADDR
```

Description

This is macro CDS_LAST_DEVICE_ADDR.

CDS_SIGNATURE Macro

File

bt_storage.h

C

```
#define CDS_SIGNATURE HIDS_SIGNATURE
```

Description

This is macro CDS_SIGNATURE.

CDS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

```
#define CDS_SIGNATURE_ADDR HIDS_SIGNATURE_ADDR
```

Description

This is macro CDS_SIGNATURE_ADDR.

FALSE Macro

File

bt_private.h

C

```
#define FALSE 0
```

Description

This is macro FALSE.

HCI_BDADDR_LEN Macro

File

bt_storage.h

C

```
#define HCI_BDADDR_LEN 6
```

Description

This is macro HCI_BDADDR_LEN.

HIDS_LAST_DEVICE_ADDR Macro

File

bt_storage.h

C

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

Description

This is macro HIDS_LAST_DEVICE_ADDR.

HIDS_SIGNATURE Macro

File

bt_storage.h

C

```
#define HIDS_SIGNATURE 0x8E
```

Description

This is macro HIDS_SIGNATURE.

HIDS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

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

Description

This is macro HIDS_SIGNATURE_ADDR.

LKS_FIRST_KEY_ADDR Macro

File

bt_storage.h

C

```
#define LKS_FIRST_KEY_ADDR 2
```

Description

This is macro LKS_FIRST_KEY_ADDR.

LKS_MAX_LINK_KEYS Macro

File

bt_storage.h

C

```
#define LKS_MAX_LINK_KEYS 5
```

Description

This is macro LKS_MAX_LINK_KEYS.

LKS_SIGNATURE Macro

File

bt_storage.h

C

```
#define LKS_SIGNATURE 0x9D
```

Description

This is macro LKS_SIGNATURE.

LKS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

```
#define LKS_SIGNATURE_ADDR 0
```

Description

This is macro LKS_SIGNATURE_ADDR.

LOG Macro

File

bt_private.h

C

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

Description

This is macro LOG.

LOG_EX Macro

File

bt_private.h

C

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

Description

This is macro LOG_EX.

LOGADDR Macro

File

bt_private.h

C

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

Description

This is macro LOGADDR.

LOGADDR_EX Macro

File

bt_private.h

C

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

Description

This is macro LOGADDR_EX.

LOGCLEAR Macro

File

bt_private.h

C

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

Description

This is macro LOGCLEAR.

LOGINT Macro

File

bt_private.h

C

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

Description

This is macro LOGINT.

LOGINT_EX Macro

File

bt_private.h

C

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

Description

This is macro LOGINT_EX.

LOGMEMORY Macro

File

bt_private.h

C

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

Description

This is macro LOGMEMORY.

LOGMEMORY_EX Macro

File

bt_private.h

C

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

Description

This is macro LOGMEMORY_EX.

LOGWRITE Macro

File

bt_private.h

C

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

Description

This is macro LOGWRITE.

NULL Macro

File

bt_private.h

C

```
#define NULL 0
```

Description

This is macro NULL.

TRUE Macro

File

bt_private.h

C

```
#define TRUE 1
```

Description

This is macro TRUE.

_bt_signal_t Structure

File

bt_signal.h

C

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

Description

This is record _bt_signal_t.

bt_bdaddr_cp Type

File

bt_types.h

C

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

Description

This is type bt_bdaddr_cp.

bt_bdaddr_p Type

File

bt_types.h

C

```
typedef bt_bdaddr_t* bt_bdaddr_p;
```

Description

This is type bt_bdaddr_p.

bt_bdaddr_t Structure

File

bt_types.h

C

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

Description

This is type bt_bdaddr_t.

bt_bool Type

File

bt_types.h

C

```
typedef bt_int bt_bool;
```

Description

This is type bt_bool.

bt_byte_cp Type

File

bt_types.h

C

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

Description

This is type bt_byte_cp.

bt_byte_p Type

File

bt_types.h

C

```
typedef bt_byte* bt_byte_p;
```

Description

This is type bt_byte_p.

bt_char Type

File

bt_types.h

C

```
typedef char bt_char;
```

Description

This is type bt_char.

bt_char_cp Type

File

bt_types.h

C

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

Description

This is type bt_char_cp.

bt_char_p Type

File

bt_types.h

C

```
typedef bt_char* bt_char_p;
```

Description

This is type bt_char_p.

bt_id Type

File

bt_types.h

C

```
typedef bt_int bt_id;
```

Description

This is type bt_id.

bt_int_cp Type

File

bt_types.h

C

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

Description

This is type bt_int_cp.

bt_int_p Type

File

bt_types.h

C

```
typedef bt_int* bt_int_p;
```

Description

This is type bt_int_p.

bt_linkkey_cp Type

File

bt_types.h

C

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

Description

This is type bt_linkkey_cp.

bt_linkkey_notification_t Structure

File

bt_oem.h

C

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

Description

This is type bt_linkkey_notification_t.

bt_linkkey_p Type

File

bt_types.h

C

```
typedef bt_linkkey_t* bt_linkkey_p;
```

Description

This is type bt_linkkey_p.

bt_linkkey_request_t Structure

File

bt_oem.h

C

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

Description

This is type bt_linkkey_request_t.

bt_linkkey_t Structure

File

bt_types.h

C

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

Description

This is type bt_linkkey_t.

bt_long_cp Type

File

bt_types.h

C

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

Description

This is type bt_long_cp.

bt_long_p Type

File

bt_types.h

C

```
typedef bt_long* bt_long_p;
```

Description

This is type bt_long_p.

bt_oem_recv_callback_fp Type

File

bt_hcitr.h

C

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

Description

- Receive callback.

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

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

bt_oem_send_callback_fp Type

File

bt_hcitr.h

C

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

Description

- Send callback.

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

bt_signal_handler_fp Type

File

bt_signal.h

C

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

Description

This is type bt_signal_handler_fp.

bt_signal_t Type

File

bt_signal.h

C

```
typedef struct _bt_signal_t bt_signal_t;
```

Description

This is type bt_signal_t.

bt_storage_callback_fp Type

File

bt_storage.h

C

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

Description

brief Storage callback.

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

bt_sys_callback_fp Type

File

bt_system.h

C

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

Description

- System start callback.

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

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

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

bt_timer_callback_fp Type

File

bt_timer.h

C

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

Description

brief Timer callback.

details This callback is called when a timer expires.

bt_timer_id Enumeration

File

bt_timer.h

C

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

Description

This is type bt_timer_id.

bt_uint_cp Type

File

bt_types.h

C

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

Description

This is type bt_uint_cp.

bt_uint_p Type

File

bt_types.h

C

```
typedef bt_uint* bt_uint_p;
```

Description

This is type bt_uint_p.

bt_ulong_cp Type

File

bt_types.h

C

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

Description

This is type bt_ulong_cp.

bt_ulong_p Type

File

bt_types.h

C

```
typedef bt_ulong* bt_ulong_p;
```

Description

This is type bt_ulong_p.

bt_uuid_cp Type

File

bt_types.h

C

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

Description

This is type bt_uuid_cp.

bt_uuid_p Type

File

bt_types.h

C

```
typedef bt_uuid_t* bt_uuid_p;
```

Description

This is type bt_uuid_p.

bt_uuid_t Structure

File

bt_types.h

C

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

Description

This is type bt_uuid_t.

bt_uuid16 Type

File

bt_types.h

C

```
typedef bt_uint bt_uuid16;
```

Description

This is type bt_uuid16.

bt_uuid32 Type

File

bt_types.h

C

```
typedef bt_ulong bt_uuid32;
```

Description

This is type bt_uuid32.

__BLUETOOTH_STG_H Macro

File

bt_storage.h

C

```
#define __BLUETOOTH_STG_H
```

Description

This is macro __BLUETOOTH_STG_H.

A2DP Functions

_bt_a2dp_avdtp_mgr_callback Function

File

a2dp_private.h

C

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

Description

This is function _bt_a2dp_avdtp_mgr_callback.

bt_a2dp_aac_codec_handler Function

File

a2dp_codec_aac.h

C

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

Description

This is function bt_a2dp_aac_codec_handler.

bt_a2dp_find_server Function

File

a2dp.h

C

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

Description

This is function bt_a2dp_find_server.

bt_a2dp_find_sink Function

File

a2dp.h

C

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

Description

brief Find sink ingroup a2dp

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

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

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

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

bt_a2dp_find_source Function

File

a2dp.h

C

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

Description

brief Find source ingroup a2dp

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

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

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

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

bt_a2dp_get_mgr Function

File

a2dp.h

C

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

Description

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

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

bt_a2dp_init Function

File

a2dp.h

C

```
void bt_a2dp_init();
```

Description

brief Initialize the A2DP layer. ingroup a2dp

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

bt_a2dp_mpeg_codec_handler Function

File

a2dp_codec_mpeg.h

C

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

Description

This is function bt_a2dp_mpeg_codec_handler.

bt_a2dp_open_and_start_stream Function

File

a2dp.h

C

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

Description

brief Open & start a stream ingroup a2dp

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

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

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

bt_a2dp_register_aac_codec Function

File

a2dp.h

C

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

Description

brief Register default AAC codec ingroup a2dp

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

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

param mgr A2DP manager.

bt_a2dp_register_callback Function

File

a2dp.h

C

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

Description

brief Register a A2DP application callback. ingroup a2dp

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

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

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

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

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

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

bt_a2dp_register_mpeg_codec Function

File

a2dp.h

C

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

Description

brief Register default MPEG codec ingroup a2dp

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

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

param mgr A2DP manager.

bt_a2dp_sbc_codec_handler Function

File

a2dp_codec_sbc.h

C

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

Description

This is function bt_a2dp_sbc_codec_handler.

bt_a2dp_start Function

File

a2dp.h

C

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

Description

brief Start the A2DP layer. ingroup a2dp

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

param mgr AVDTP manager.

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

bt_a2dp_set_configuration Function

File

a2dp.h

C

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

Description

This is function bt_a2dp_set_configuration.

A2DP Data Types and Constants

_bt_a2dp_event_u Union

File

a2dp.h

C

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

Members

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

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

Description

brief Parameter to an application callback. ingroup a2dp

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

_bt_a2dp_mgr_t Structure

File

a2dp.h

C

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

Members

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

Description

brief A2DP manager. ingroup a2dp

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

bt_a2dp_aac_config_t Structure

File

a2dp_codec_aac.h

C

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

Description

This is type bt_a2dp_aac_config_t.

bt_a2dp_event_t Type

File

a2dp.h

C

```
typedef union _bt_a2dp_event_u bt_a2dp_event_t;
```

Description

This is type bt_a2dp_event_t.

bt_a2dp_evt_open_and_start_stream_completed_t Structure

File

a2dp.h

C

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

Members

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

Description

brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp

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

bt_a2dp_find_server_callback_fp Type

File

a2dp.h

C

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

Description

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

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

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

bt_a2dp_mgr_callback_fp Type

File

a2dp.h

C

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

Description

brief A2DP application callback. ingroup a2dp

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

param mgr A2DP manager.

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

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

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

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

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

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

@arg A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED c bt_a2dp_evt_open_and_start_stream_completed_t open_and_start_stream_completed

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

bt_a2dp_mgr_t Type

File

a2dp.h

C

```
typedef struct _bt_a2dp_mgr_t bt_a2dp_mgr_t;
```

Description

This is type bt_a2dp_mgr_t.

bt_a2dp_mpeg_config_t Structure

File

a2dp_codec_mpeg.h

C

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

Description

This is type bt_a2dp_mpeg_config_t.

bt_a2dp_sbc_config_t Structure

File

a2dp_codec_sbc.h

C

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

Description

This is type bt_a2dp_sbc_config_t.

bt_a2dp_sbc_packet_info_t Structure

File

a2dp_codec_sbc.h

C

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

Description

This is type bt_a2dp_sbc_packet_info_t.

__A2DP_CODEC_AAC_H Macro

File

a2dp_codec_aac.h

C

```
#define __A2DP_CODEC_AAC_H
```

Description

This is macro __A2DP_CODEC_AAC_H.

__A2DP_CODEC_MPEG_H Macro

File

a2dp_codec_mpeg.h

C

```
#define __A2DP_CODEC_MPEG_H
```

Description

This is macro __A2DP_CODEC_MPEG_H.

__A2DP_CODEC_SBC_H Macro

File

a2dp_codec_sbc.h

C

```
#define __A2DP_CODEC_SBC_H
```

Description

This is macro __A2DP_CODEC_SBC_H.

__A2DP_H Macro

File

a2dp.h

C

```
#define __A2DP_H
```

Description

This is macro __A2DP_H.

__A2DP_PRIVATE_H Macro

File

a2dp_private.h

C

```
#define __A2DP_PRIVATE_H
```

Description

This is macro __A2DP_PRIVATE_H.

A2DP_EVT_ABORT_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_ABORT_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_CLOSE_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_CLOSE_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_CTRL_CHANNEL_CONNECTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_CTRL_CHANNEL_DISCONNECTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_CTRL_CONNECTION_FAILED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_DISCOVER_SEP_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED Macro

File

a2dp.h

C

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

```
capabilities" request.
```

Description

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

A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_MEDIA_PACKET_RECEIVED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_MEDIA_PACKET_SEND_FAILED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_MEDIA_PACKET_SENT Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_NOTHING Macro

File

a2dp.h

C

```
#define A2DP_EVT_NOTHING AVDTP_EVT_NULL
```

Description

This is macro A2DP_EVT_NOTHING.

A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_OPEN_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_OPEN_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_RECONFIGURE_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_RECONFIGURE_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SEP_CAPABILITIES_RECEIVED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SEP_INFO_RECEIVED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_START_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_START_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_ABORTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_CLOSED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_CONFIGURATION_RECEIVED Macro

File

a2dp.h

C

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

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

Description

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

A2DP_EVT_STREAM_CONFIGURED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_OPENED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_RECONFIGURED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_STARTED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_STREAM_SUSPENDED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SUSPEND_STREAM_COMPLETED Macro

File

a2dp.h

C

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

Description

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

A2DP_EVT_SUSPEND_STREAM_REQUESTED Macro

File

a2dp.h

C

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

Description

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

A2DP_MANAGER_STATE_CONNECTING Macro

File

a2dp.h

C

```
#define A2DP_MANAGER_STATE_CONNECTING 1
```

Description

This is macro A2DP_MANAGER_STATE_CONNECTING.

A2DP_MANAGER_STATE_IDLE Macro

File

a2dp.h

C

```
#define A2DP_MANAGER_STATE_IDLE 0
```

Description

This is macro A2DP_MANAGER_STATE_IDLE.

A2DP_SINK_FEATURE_AMPLIFIER Macro

File

a2dp.h

C

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

Description

< Amplifier

A2DP_SINK_FEATURE_HEADPHONE Macro

File

a2dp.h

C

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

Description

< Headphone

A2DP_SINK_FEATURE_RECORDER Macro

File

a2dp.h

C

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

Description

< Recorder

A2DP_SINK_FEATURE_SPEAKER Macro

File

a2dp.h

C

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

Description

< Speaker

A2DP_SOURCE_FEATURE_MICROPHONE Macro

File

a2dp.h

C

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

Description

< Mic

A2DP_SOURCE_FEATURE_MIXER Macro

File

a2dp.h

C

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

Description

< Mixer

A2DP_SOURCE_FEATURE_PLAYER Macro

File

a2dp.h

C

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

Description

< Player

A2DP_SOURCE_FEATURE_TUNER Macro

File

a2dp.h

C

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

Description

< Tuner

AAC_CHANNELS_1 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_1 0x2
```

Description

This is macro AAC_CHANNELS_1.

AAC_CHANNELS_2 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_2 0x1
```

Description

This is macro AAC_CHANNELS_2.

AAC_CHANNELS_ALL Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_ALL 0x3
```

Description

This is macro AAC_CHANNELS_ALL.

AAC_OBJECT_TYPE_MPEG_2_LC Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_2_LC 0x80
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_2_LC.

AAC_OBJECT_TYPE_MPEG_4_LC Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_LC 0x40
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_LC.

AAC_OBJECT_TYPE_MPEG_4_LTP Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_LTP 0x20
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.

AAC_OBJECT_TYPE_MPEG_4_SCALABLE Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_SCALABLE 0x10
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.

AAC_SAMPLING_FREQUENCY_11025 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_11025 0x040
```

Description

This is macro AAC_SAMPLING_FREQUENCY_11025.

AAC_SAMPLING_FREQUENCY_12000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_12000 0x020
```

Description

This is macro AAC_SAMPLING_FREQUENCY_12000.

AAC_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_16000 0x010
```

Description

This is macro AAC_SAMPLING_FREQUENCY_16000.

AAC_SAMPLING_FREQUENCY_22050 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_22050 0x008
```

Description

This is macro AAC_SAMPLING_FREQUENCY_22050.

AAC_SAMPLING_FREQUENCY_24000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_24000 0x004
```

Description

This is macro AAC_SAMPLING_FREQUENCY_24000.

AAC_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_32000 0x002
```

Description

This is macro AAC_SAMPLING_FREQUENCY_32000.

AAC_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_44100 0x001
```

Description

This is macro AAC_SAMPLING_FREQUENCY_44100.

AAC_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_48000 0x800
```

Description

This is macro AAC_SAMPLING_FREQUENCY_48000.

AAC_SAMPLING_FREQUENCY_64000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_64000 0x400
```

Description

This is macro AAC_SAMPLING_FREQUENCY_64000.

AAC_SAMPLING_FREQUENCY_8000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_8000 0x080
```

Description

This is macro AAC_SAMPLING_FREQUENCY_8000.

AAC_SAMPLING_FREQUENCY_88200 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_88200 0x200
```

Description

This is macro AAC_SAMPLING_FREQUENCY_88200.

AAC_SAMPLING_FREQUENCY_96000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_96000 0x100
```

Description

This is macro AAC_SAMPLING_FREQUENCY_96000.

AAC_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_ALL 0xFFF
```

Description

This is macro AAC_SAMPLING_FREQUENCY_ALL.

AAC_VBR_NOT_SUPPORTED Macro

File

a2dp_codec_aac.h

C

```
#define AAC_VBR_NOT_SUPPORTED 0
```

Description

This is macro AAC_VBR_NOT_SUPPORTED.

AAC_VBR_SUPPORTED Macro

File

a2dp_codec_aac.h

C

```
#define AAC_VBR_SUPPORTED 1
```

Description

This is macro AAC_VBR_SUPPORTED.

bt_a2dp_abort_stream Macro

File

a2dp.h

C

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

Description

brief Suspend a stream. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_add_media_rx_buffer Macro

File

a2dp.h

C

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

Description

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

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

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

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

bt_a2dp_add_media_tx_buffer Macro

File

a2dp.h

C

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

Description

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

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

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

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

bt_a2dp_call_codec Macro

File

a2dp.h

C

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

Description

This is macro bt_a2dp_call_codec.

bt_a2dp_cancel_listen Macro

File

a2dp.h

C

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

Description

brief Cancel listening for incoming connections. ingroup a2dp

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

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

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

bt_a2dp_close_stream Macro

File

a2dp.h

C

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

Description

brief Close a stream. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_connect Macro

File

a2dp.h

C

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

Description

brief Connect to a remote device. ingroup a2dp

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

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

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

bt_a2dp_connect_ex Macro

File

a2dp.h

C

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

Description

brief Connect to a remote device. ingroup a2dp

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

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

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

bt_a2dp_create_stream Macro

File

a2dp.h

C

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

Description

brief Create a stream. ingroup a2dp

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

param mgr A2DP manager.

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

bt_a2dp_destroy_stream Macro

File

a2dp.h

C

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

Description

brief Destroy a stream. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_disconnect Macro

File

a2dp.h

C

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

Description

brief Disconnect from a remote device. ingroup a2dp

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

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

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

bt_a2dp_discover Macro

File

a2dp.h

C

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

Description

brief Discover SEPs on a remote device. ingroup a2dp

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

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

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

bt_a2dp_find_codec Macro

File

a2dp.h

C

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

Description

brief Find a codec ingroup a2dp

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

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

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

bt_a2dp_get_capabilities Macro

File

a2dp.h

C

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

Description

brief Get remote SEP capabilities. ingroup a2dp

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

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

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

bt_a2dp_get_hci_connection Macro

File

a2dp.h

C

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

Description

brief Get HCI connection for a stream ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

note This function has not been implemented.

bt_a2dp_get_stream_codec_config Macro

File

a2dp.h

C

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

Description

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

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

param mgr A2DP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_a2dp_get_stream_codec_type Macro

File

a2dp.h

C

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

Description

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

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

param mgr A2DP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

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

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

@arg 0xFF otherwise.

bt_a2dp_get_stream_local_sep_id Macro

File

a2dp.h

C

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

Description

brief Get stream's local SEP ID. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_get_stream_remote_address Macro

File

a2dp.h

C

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

Description

brief Get stream's remote BT address. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_get_stream_remote_sep_id Macro

File

a2dp.h

C

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

Description

brief Get stream's remote SEP ID. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_get_stream_state Macro

File

a2dp.h

C

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

Description

brief Get local stream state. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_listen Macro

File

a2dp.h

C

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

Description

brief Listen for incoming connections. ingroup a2dp

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

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

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

bt_a2dp_reconfigure_stream Macro

File

a2dp.h

C

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

Description

brief Reconfigure stream. ingroup a2dp

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

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

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

bt_a2dp_register_sink Macro

File

a2dp.h

C

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

Description

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

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

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

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

bt_a2dp_register_source Macro

File

a2dp.h

C

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

Description

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

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

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

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

bt_a2dp_remove_media_rx_buffer Macro

File

a2dp.h

C

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

Description

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

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

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

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

bt_a2dp_remove_media_tx_buffer Macro

File

a2dp.h

C

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

Description

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

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

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

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

note This function has not been implemented.

bt_a2dp_start_stream Macro

File

a2dp.h

C

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

Description

brief Start a stream. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_suspend_stream Macro

File

a2dp.h

C

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

Description

brief Suspend a stream. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

MPEG_BITRATE_0000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0000 0x0080
```

Description

This is macro MPEG_BITRATE_0000.

MPEG_BITRATE_0001 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0001 0x0100
```

Description

This is macro MPEG_BITRATE_0001.

MPEG_BITRATE_0010 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0010 0x0200
```

Description

This is macro MPEG_BITRATE_0010.

MPEG_BITRATE_0011 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0011 0x0400
```

Description

This is macro MPEG_BITRATE_0011.

MPEG_BITRATE_0100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0100 0x0800
```

Description

This is macro MPEG_BITRATE_0100.

MPEG_BITRATE_0101 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0101 0x1000
```

Description

This is macro MPEG_BITRATE_0101.

MPEG_BITRATE_0110 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0110 0x2000
```

Description

This is macro MPEG_BITRATE_0110.

MPEG_BITRATE_0111 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0111 0x4000
```

Description

This is macro MPEG_BITRATE_0111.

MPEG_BITRATE_1000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1000 0x0001
```

Description

This is macro MPEG_BITRATE_1000.

MPEG_BITRATE_1001 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1001 0x0002
```

Description

This is macro MPEG_BITRATE_1001.

MPEG_BITRATE_1010 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1010 0x0004
```

Description

This is macro MPEG_BITRATE_1010.

MPEG_BITRATE_1011 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1011 0x0008
```

Description

This is macro MPEG_BITRATE_1011.

MPEG_BITRATE_1100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1100 0x0010
```

Description

This is macro MPEG_BITRATE_1100.

MPEG_BITRATE_1101 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1101 0x0020
```

Description

This is macro MPEG_BITRATE_1101.

MPEG_BITRATE_1110 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1110 0x0040
```

Description

This is macro MPEG_BITRATE_1110.

MPEG_BITRATE_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_ALL 0x7FFF
```

Description

This is macro MPEG_BITRATE_ALL.

MPEG_CHANNEL_MODE_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_ALL 15
```

Description

This is macro MPEG_CHANNEL_MODE_ALL.

MPEG_CHANNEL_MODE_DUAL_CHANNEL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_DUAL_CHANNEL 4
```

Description

This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.

MPEG_CHANNEL_MODE_JOINT_STEREO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_JOINT_STEREO 1
```

Description

This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.

MPEG_CHANNEL_MODE_MONO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_MONO 8
```

Description

This is macro MPEG_CHANNEL_MODE_MONO.

MPEG_CHANNEL_MODE_STEREO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_STEREO 2
```

Description

This is macro MPEG_CHANNEL_MODE_STEREO.

MPEG_CRC_PROTECTION_NOT_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CRC_PROTECTION_NOT_SUPPORTED 0
```

Description

This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.

MPEG_CRC_PROTECTION_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CRC_PROTECTION_SUPPORTED 1
```

Description

This is macro MPEG_CRC_PROTECTION_SUPPORTED.

MPEG_LAYER_1 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_1 0x04
```

Description

This is macro MPEG_LAYER_1.

MPEG_LAYER_2 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_2 0x02
```

Description

This is macro MPEG_LAYER_2.

MPEG_LAYER_3 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_3 0x01
```

Description

This is macro MPEG_LAYER_3.

MPEG_LAYER_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_ALL 0x07
```

Description

This is macro MPEG_LAYER_ALL.

MPEG_MPF_1 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_MPF_1 0
```

Description

This is macro MPEG_MPF_1.

MPEG_MPF_2 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_MPF_2 1
```

Description

This is macro MPEG_MPF_2.

MPEG_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_16000 0x20
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_16000.

MPEG_SAMPLING_FREQUENCY_22050 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_22050 0x10
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_22050.

MPEG_SAMPLING_FREQUENCY_24000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_24000 0x08
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_24000.

MPEG_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_32000 0x04
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_32000.

MPEG_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_44100 0x02
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_44100.

MPEG_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_48000 0x01
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_48000.

MPEG_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_ALL 0x3F
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_ALL.

MPEG_VBR_NOT_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_VBR_NOT_SUPPORTED 0
```

Description

This is macro MPEG_VBR_NOT_SUPPORTED.

MPEG_VBR_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_VBR_SUPPORTED 1
```

Description

This is macro MPEG_VBR_SUPPORTED.

SBC_ALLOCATION_METHOD_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_ALL 3
```

Description

This is macro SBC_ALLOCATION_METHOD_ALL.

SBC_ALLOCATION_METHOD_LOUDNESS Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_LOUDNESS 1
```

Description

This is macro SBC_ALLOCATION_METHOD_LOUDNESS.

SBC_ALLOCATION_METHOD_SNR Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_SNR 2
```

Description

This is macro SBC_ALLOCATION_METHOD_SNR.

SBC_BLOCK_LENGTH_12 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_12 2
```

Description

This is macro SBC_BLOCK_LENGTH_12.

SBC_BLOCK_LENGTH_16 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_16 1
```

Description

This is macro SBC_BLOCK_LENGTH_16.

SBC_BLOCK_LENGTH_4 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_4 8
```

Description

This is macro SBC_BLOCK_LENGTH_4.

SBC_BLOCK_LENGTH_8 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_8 4
```

Description

This is macro SBC_BLOCK_LENGTH_8.

SBC_BLOCK_LENGTH_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_ALL 15
```

Description

This is macro SBC_BLOCK_LENGTH_ALL.

SBC_CHANNEL_MODE_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_ALL 15
```

Description

This is macro SBC_CHANNEL_MODE_ALL.

SBC_CHANNEL_MODE_DUAL_CHANNEL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_DUAL_CHANNEL 4
```

Description

This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.

SBC_CHANNEL_MODE_JOINT_STEREO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_JOINT_STEREO 1
```

Description

This is macro SBC_CHANNEL_MODE_JOINT_STEREO.

SBC_CHANNEL_MODE_MONO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_MONO 8
```

Description

This is macro SBC_CHANNEL_MODE_MONO.

SBC_CHANNEL_MODE_STEREO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_STEREO 2
```

Description

This is macro SBC_CHANNEL_MODE_STEREO.

SBC_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_16000 8
```

Description

This is macro SBC_SAMPLING_FREQUENCY_16000.

SBC_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_32000 4
```

Description

This is macro SBC_SAMPLING_FREQUENCY_32000.

SBC_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_44100 2
```

Description

This is macro SBC_SAMPLING_FREQUENCY_44100.

SBC_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_48000 1
```

Description

This is macro SBC_SAMPLING_FREQUENCY_48000.

SBC_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_ALL 15
```

Description

This is macro SBC_SAMPLING_FREQUENCY_ALL.

SBC_SUBBANDS_4 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_4 2
```

Description

This is macro SBC_SUBBANDS_4.

SBC_SUBBANDS_8 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_8 1
```

Description

This is macro SBC_SUBBANDS_8.

SBC_SUBBANDS_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_ALL 3
```

Description

This is macro SBC_SUBBANDS_ALL.

bt_a2dp_get_all_capabilities Macro

File

a2dp.h

C

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

Description

brief Get remote SEP capabilities. ingroup a2dp

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

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

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

bt_a2dp_get_configuration Macro

File

a2dp.h

C

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

Description

This is macro bt_a2dp_get_configuration.

bt_a2dp_open_stream Macro

File

a2dp.h

C

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

Description

This is macro bt_a2dp_open_stream.

A2DP_EVT_SET_STREAM_CONFIGURATION Macro

File

a2dp.h

C

```
#define A2DP_EVT_SET_STREAM_CONFIGURATION AVDTP_EVT_SET_STREAM_CONFIGURATION
```

Description

This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.

bt_a2dp_clear_media_tx_queue Macro

File

a2dp.h

C

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

Description

brief Clear send queue ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

bt_a2dp_get_stream_config Macro

File

a2dp.h

C

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

Description

brief Get stream's configuration. ingroup a2dp

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

param mgr A2DP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_a2dp_get_stream_direction Macro

File

a2dp.h

C

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

Description

This is macro bt_a2dp_get_stream_direction.

bt_a2dp_report_delay Macro

File

a2dp.h

C

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

Description

This is macro bt_a2dp_report_delay.

bt_a2dp_set_media_tx_queue_limit Macro

File

a2dp.h

C

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

Description

brief Set limit on the send queue ingroup a2dp

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

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

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

AVCTP Functions

_bt_avctp_allocate_channel Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_allocate_channel.

_bt_avctp_allocate_message Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_allocate_message.

_bt_avctp_allocate_transport Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_allocate_transport.

_bt_avctp_find_channel Function

File

avctp_private.h

C

```
bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
```

```
profile_id, bt_int psm);
```

Description

This is function _bt_avctp_find_channel.

_bt_avctp_find_transport Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_find_transport.

_bt_avctp_free_channel Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_free_channel.

_bt_avctp_free_message Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_free_message.

_bt_avctp_free_transport Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_free_transport.

_bt_avctp_init_message_buffers Function

File

avctp_private.h

C

```
void _bt_avctp_init_message_buffers();
```

Description

This is function _bt_avctp_init_message_buffers.

_bt_avctp_init_signal Function

File

avctp_private.h

C

```
void _bt_avctp_init_signal();
```

Description

This is function _bt_avctp_init_signal.

_bt_avctp_l2cap_read_data_callback Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_l2cap_read_data_callback.

_bt_avctp_packet_assembler Function

File

avctp_packet.h

C

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

Description

This is function _bt_avctp_packet_assembler.

_bt_avctp_send_ipid Function

File

avctp_private.h

C

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

Description

This is function _bt_avctp_send_ipid.

_bt_avctp_set_signal Function

File

avctp_private.h

C

```
void _bt_avctp_set_signal();
```

Description

This is function _bt_avctp_set_signal.

_bt_avrcp_init_cmd_buffers Function

File

avctp_private.h

C

```
void _bt_avrcp_init_cmd_buffers();
```

Description

This is function _bt_avrcp_init_cmd_buffers.

bt_avctp_cancel_command Function

File

avctp.h

C

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

Description

brief Cancel a command message. ingroup avctp

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

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

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

bt_avctp_cancel_listen Function

File

avctp.h

C

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

Description

brief Cancel listening for incoming connections. ingroup avctp

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

param channel AVCTP channel.

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

bt_avctp_cancel_response Function

File

avctp.h

C

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

Description

brief Cancel a response message. ingroup avctp

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

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

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

bt_avctp_connect Function

File

avctp.h

C

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

Description

brief Connect to a remote device. ingroup avctp

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

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

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

bt_avctp_create_channel Function

File

avctp.h

C

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

Description

brief Allocate AVCTP channel ingroup avctp

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

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

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

bt_avctp_create_outgoing_channel Function

File

avctp.h

C

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

Description

brief Allocate AVCTP channel ingroup avctp

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

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

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

bt_avctp_destroy_channel Function

File

avctp.h

C

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

Description

brief Destroy AVCTP channel. ingroup avctp

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

param channel AVCTP channel.

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

bt_avctp_disconnect Function

File

avctp.h

C

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

Description

brief Disconnect from a remote device. ingroup avctp

details This function closes a connection to a remote device.

param channel AVCTP channel.

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

bt_avctp_get_channel_remote_address Function

File

avctp.h

C

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

Description

brief Get channel's remote BT address. ingroup avctp

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

param channel AVCTP channel.

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

bt_avctp_get_channel_state Function

File

avctp.h

C

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

Description

brief Get channel state ingroup avctp

details This function return current state of the specified channel

param channel AVCTP channel.

return li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avctp_get_hci_connection Function

File

avctp.h

C

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

Description

brief Get HCI connection for a channel ingroup avctp

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

param channel AVCTP channel.

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

bt_avctp_get_mgr Function

File

avctp.h

C

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

Description

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

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

bt_avctp_init Function

File

avctp.h

C

```
void bt_avctp_init();
```

Description

brief Initialize the AVCTP layer. ingroup avctp

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

bt_avctp_listen Function

File

avctp.h

C

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

Description

brief Listen for incoming connections. ingroup avctp

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

param channel AVCTP channel.

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

bt_avctp_send_command Function

File

avctp.h

C

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

Description

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

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

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

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

bt_avctp_send_response Function

File

avctp.h

C

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

Description

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

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

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

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

bt_avctp_set_callback Function

File

avctp.h

C

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

Description

brief Register a AVCTP application callback. ingroup avctp

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

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

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

bt_avctp_start Function

File

avctp.h

C

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

Description

This is function bt_avctp_start.

AVCTP Data Types and Constants

_bt_avctp_channel_t Structure

File

avctp.h

C

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

Description

brief AVCTP channel description ingroup avctp

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

_bt_avctp_message_t Structure

File

avctp.h

C

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

Description

brief AVCTP message description ingroup avctp

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

_bt_avctp_mgr_t Structure

File

avctp.h

C

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

Members

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

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

Description

brief AVCTP manager. ingroup avctp

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

_bt_avctp_transport_t Structure

File

avctp.h

C

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

Description

brief AVCTP transport description ingroup avctp

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

bt_avctp_channel_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_channel_t bt_avctp_channel_t;
```

Description

This is type bt_avctp_channel_t.

bt_avctp_event_t Union

File

avctp.h

C

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

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

Members

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

Description

brief Parameter to an application callback. ingroup avctp

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

bt_avctp_evt_channel_connected_t Structure

File

avctp.h

C

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

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp

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

bt_avctp_evt_channel_disconnected_t Structure

File

avctp.h

C

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

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp

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

bt_avctp_evt_command_cancelled_t Structure

File

avctp.h

C

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

Members

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

Description

brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp

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

bt_avctp_evt_command_received_t Structure

File

avctp.h

C

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

Members

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

Description

brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp

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

bt_avctp_evt_command_sent_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_command_sent_t {
  bt_avctp_channel_t* channel;
```

```
  bt_avctp_message_t* command;
} bt_avctp_evt_command_sent_t;
```

Members

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

Description

brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp

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

bt_avctp_evt_connection_failed_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_connection_failed_t {
  bt_avctp_channel_t* channel;
} bt_avctp_evt_connection_failed_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.

bt_avctp_evt_response_cancelled_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_cancelled_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_cancelled_t;
```

Members

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

Description

brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp

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

bt_avctp_evt_response_received_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_received_t {
```

```
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_received_t;
```

Members

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

Description

brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp

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

bt_avctp_evt_response_sent_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_sent_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_sent_t;
```

Members

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

Description

brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.

bt_avctp_message_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_message_t bt_avctp_message_t;
```

Description

This is type bt_avctp_message_t.

bt_avctp_mgr_callback_fp Type

File

avctp.h

C

```
typedef void (* bt_avctp_mgr_callback_fp)(struct _bt_avctp_mgr_t* mgr, bt_byte evt, bt_avctp_event_t*
evt_param, void* callback_param);
```

Description

brief AVCTP application callback. ingroup avctp

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

param mgr AVCTP manager.

param evt AVCTP event. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device).

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

param evt_param Event parameter. Which member of the bt_avctp_event_t union is valid depends on the event: @arg AVCTP_EVT_CHANNEL_CONNECTED c bt_avctp_evt_channel_connected_t channel_connected @arg AVCTP_EVT_CHANNEL_DISCONNECTED c bt_avctp_evt_channel_disconnected_t channel_disconnected @arg AVCTP_EVT_CONNECTION_FAILED c bt_avctp_evt_connection_failed_t connection_failed @arg AVCTP_EVT_COMMAND_RECEIVED c bt_avctp_evt_command_received_t command_received @arg AVCTP_EVT_RESPONE_RECEIVED c bt_avctp_evt_response_received_t response_received @arg AVCTP_EVT_COMMAND_SENT c bt_avctp_evt_command_sent_t command_sent @arg AVCTP_EVT_RESPONSE_SENT c bt_avctp_evt_response_sent_t response_sent @arg AVCTP_EVT_COMMAND_CANCELLED c bt_avctp_evt_command_cancelled_t command_cancelled @arg AVCTP_EVT_RESPONSE_CANCELLED c bt_avctp_evt_response_cancelled_t response_cancelled

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

bt_avctp_mgr_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_mgr_t bt_avctp_mgr_t;
```

Description

This is type bt_avctp_mgr_t.

bt_avctp_packet_t Structure

File

avctp_packet.h

C

```
typedef struct _bt_avctp_packet_t {
  bt_packet_t header;
  bt_byte tran_id;
  bt_byte packet_type;
  bt_byte num_packets;
  bt_byte message_type;
  bt_byte ipid;
  bt_uint profile_id;
  bt_byte* data;
  bt_uint data_len;
  bt_int data_pos;
} bt_avctp_packet_t;
```

Description

This is type bt_avctp_packet_t.

bt_avctp_transport_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_transport_t bt_avctp_transport_t;
```

Description

This is type bt_avctp_transport_t.

__AVCTP_H Macro

File

avctp.h

C

```
#define __AVCTP_H
```

Description

This is macro __AVCTP_H.

__AVCTP_PACKET_H Macro

File

avctp_packet.h

C

```
#define __AVCTP_PACKET_H
```

Description

This is macro __AVCTP_PACKET_H.

__AVCTP_PRIVATE_H Macro

File

avctp_private.h

C

```
#define __AVCTP_PRIVATE_H
```

Description

This is macro __AVCTP_PRIVATE_H.

AVCTP_CHANNEL_FLAG_LISTENING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_FLAG_LISTENING 1
```

Description

This is macro AVCTP_CHANNEL_FLAG_LISTENING.

AVCTP_CHANNEL_FLAG_SENDING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_FLAG_SENDING 2
```

Description

This is macro AVCTP_CHANNEL_FLAG_SENDING.

AVCTP_CHANNEL_STATE_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_CONNECTED 3
```

Description

This is macro AVCTP_CHANNEL_STATE_CONNECTED.

AVCTP_CHANNEL_STATE_CONNECTING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVCTP_CHANNEL_STATE_CONNECTING.

AVCTP_CHANNEL_STATE_DISCONNECTING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_DISCONNECTING 4
```

Description

This is macro AVCTP_CHANNEL_STATE_DISCONNECTING.

AVCTP_CHANNEL_STATE_FREE Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_FREE 0
```

Description

This is macro AVCTP_CHANNEL_STATE_FREE.

AVCTP_CHANNEL_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_IDLE 1
```

Description

This is macro AVCTP_CHANNEL_STATE_IDLE.

AVCTP_ERROR_BAD_STATE Macro

File

avrcp.h

C

```
#define AVCTP_ERROR_BAD_STATE 0x31
```

Description

This is macro AVCTP_ERROR_BAD_STATE.

AVCTP_ERROR_SUCCESS Macro

File

avrcp.h

C

```
#define AVCTP_ERROR_SUCCESS 0
```

Description

This is macro AVCTP_ERROR_SUCCESS.

AVCTP_EVT_CHANNEL_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CHANNEL_CONNECTED 1   ///< This event is generated when a channel between two AVCTP
entities has been established.
```

Description

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

AVCTP_EVT_CHANNEL_DISCONNECTED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CHANNEL_DISCONNECTED 2   ///< This event is generated when a channel between two AVCTP
entities has been terminated.
```

Description

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

AVCTP_EVT_COMMAND_CANCELLED Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_CANCELLED 54  ///< This event is generated when a command has been canceled.
```

Description

< This event is generated when a command has been canceled.

AVCTP_EVT_COMMAND_RECEIVED Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_RECEIVED 50  ///< This event is generated when a local device received a command.
```

Description

< This event is generated when a local device received a command.

AVCTP_EVT_COMMAND_SENT Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_SENT 52  ///< This event is generated when a local device finished sending a
command.
```

Description

< This event is generated when a local device finished sending a command.

AVCTP_EVT_CONNECTION_FAILED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CONNECTION_FAILED 3   ///< This event is generated when a local device failed to create a
channel between two AVCTP entities.
```

Description

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

AVCTP_EVT_NOTHING Macro

File

avctp.h

C

```
#define AVCTP_EVT_NOTHING 0
```

Description

addtogroup avrcp @{

@name Events

details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

AVCTP_EVT_RESPONE_RECEIVED Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONE_RECEIVED 51  ///< This event is generated when a local device received a response.
```

Description

< This event is generated when a local device received a response.

AVCTP_EVT_RESPONSE_CANCELLED Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONSE_CANCELLED 55  ///< This event is generated when a response has been canceled.
```

Description

< This event is generated when a response has been canceled.

AVCTP_EVT_RESPONSE_SENT Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONSE_SENT 53  ///< This event is generated when a local device finished sending a
response.
```

Description

< This event is generated when a local device finished sending a response.

AVCTP_MANAGER_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVCTP_MANAGER_STATE_IDLE.

AVCTP_MESSAGE_PACKET_TYPE_CONTINUE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_CONTINUE 2
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_CONTINUE.

AVCTP_MESSAGE_PACKET_TYPE_END Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_END 3
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_END.

AVCTP_MESSAGE_PACKET_TYPE_SINGLE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_SINGLE 0
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_SINGLE.

AVCTP_MESSAGE_PACKET_TYPE_START Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_START 1
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_START.

AVCTP_MESSAGE_TYPE_COMMAND Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_TYPE_COMMAND 0
```

Description

This is macro AVCTP_MESSAGE_TYPE_COMMAND.

AVCTP_MESSAGE_TYPE_RESPONSE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_TYPE_RESPONSE 1
```

Description

This is macro AVCTP_MESSAGE_TYPE_RESPONSE.

AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED 4
```

Description

This is macro AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED.

AVCTP_TRANSPORT_FLAG_SENDING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_FLAG_SENDING 2
```

Description

This is macro AVCTP_TRANSPORT_FLAG_SENDING.

AVCTP_TRANSPORT_STATE_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_CONNECTED 3
```

Description

This is macro AVCTP_TRANSPORT_STATE_CONNECTED.

AVCTP_TRANSPORT_STATE_CONNECTING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_CONNECTING 2
```

Description

This is macro AVCTP_TRANSPORT_STATE_CONNECTING.

AVCTP_TRANSPORT_STATE_DISCONNECTING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_DISCONNECTING 4
```

Description

This is macro AVCTP_TRANSPORT_STATE_DISCONNECTING.

AVCTP_TRANSPORT_STATE_FREE Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_FREE 0
```

Description

This is macro AVCTP_TRANSPORT_STATE_FREE.

AVCTP_TRANSPORT_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_IDLE 1
```

Description

This is macro AVCTP_TRANSPORT_STATE_IDLE.

AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET 1
```

Description

This is macro AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET.

AVDTP Functions

_bt_avdtp_add_param_byte Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_byte(bt_avdtp_control_cmd_t* command, bt_byte value);
```

Description

This is function _bt_avdtp_add_param_byte.

_bt_avdtp_add_param_uint Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_uint(bt_avdtp_control_cmd_t* command, bt_uint value);
```

Description

This is function _bt_avdtp_add_param_uint.

_bt_avdtp_allocate_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_allocate_buffers();
```

Description

This is function _bt_avdtp_allocate_buffers.

_bt_avdtp_allocate_cmd Function

File

avdtp_private.h

C

```
bt_avdtp_control_cmd_t* _bt_avdtp_allocate_cmd(bt_byte msg_type);
```

Description

This is function _bt_avdtp_allocate_cmd.

_bt_avdtp_allocate_sep Function

File

avdtp_private.h

C

```
bt_avdtp_sep_t* _bt_avdtp_allocate_sep();
```

Description

This is function _bt_avdtp_allocate_sep.

_bt_avdtp_allocate_sep_config Function

File

avdtp_private.h

C

```
bt_avdtp_sep_capabilities_t* _bt_avdtp_allocate_sep_config();
```

Description

This is function _bt_avdtp_allocate_sep_config.

_bt_avdtp_allocate_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_allocate_stream(bt_avdtp_mgr_t* mgr);
```

Description

This is function _bt_avdtp_allocate_stream.

_bt_avdtp_allocate_transport_channel Function

File

avdtp_private.h

C

```
bt_avdtp_transport_channel_t* _bt_avdtp_allocate_transport_channel(bt_avdtp_mgr_t* mgr, bt_byte
strm_handle, bt_byte tc_id, bt_byte tc_type);
```

Description

This is function _bt_avdtp_allocate_transport_channel.

_bt_avdtp_allocate_transport_session_id Function

File

avdtp_private.h

C

```
bt_byte _bt_avdtp_allocate_transport_session_id(bt_avdtp_mgr_t* mgr, bt_byte ts_id);
```

Description

This is function _bt_avdtp_allocate_transport_session_id.

_bt_avdtp_begin_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_begin_tc_channel_operation(bt_byte opcode, bt_avdtp_transport_op_callback_fp callback, void*
callback_param);
```

Description

This is function _bt_avdtp_begin_tc_channel_operation.

_bt_avdtp_commit_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_commit_tc_channel_operation();
```

Description

This is function _bt_avdtp_commit_tc_channel_operation.

_bt_avdtp_control_channel_accept_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_accept_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code,
bt_byte trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_accept_handler.

_bt_avdtp_control_channel_cmd_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_cmd_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code, bt_byte
trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_cmd_handler.

_bt_avdtp_control_channel_event_handler Function

File

avdtp_control.h

C

```
void _bt_avdtp_control_channel_event_handler(bt_avdtp_control_channel_t* channel, bt_byte evt, void*
evt_param);
```

Description

This is function _bt_avdtp_control_channel_event_handler.

_bt_avdtp_control_channel_reject_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_reject_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code,
bt_byte trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_reject_handler.

_bt_avdtp_execute_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_execute_tc_channel_operation();
```

Description

This is function _bt_avdtp_execute_tc_channel_operation.

_bt_avdtp_find_listening_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_listening_stream(bt_avdtp_mgr_t* mgr, bt_byte sep_id);
```

Description

This is function _bt_avdtp_find_listening_stream.

_bt_avdtp_find_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

This is function _bt_avdtp_find_stream.

_bt_avdtp_find_stream_by_remote_sep_id Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream_by_remote_sep_id(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr,
bt_byte remote_seid);
```

Description

This is function _bt_avdtp_find_stream_by_remote_sep_id.

_bt_avdtp_find_stream_by_sep_id Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream_by_sep_id(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte
sep_id);
```

Description

This is function _bt_avdtp_find_stream_by_sep_id.

_bt_avdtp_free_cmd Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_cmd(bt_avdtp_control_cmd_t* command);
```

Description

This is function _bt_avdtp_free_cmd.

_bt_avdtp_free_sep_config Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_sep_config(bt_avdtp_sep_capabilities_t* sep_config);
```

Description

This is function _bt_avdtp_free_sep_config.

_bt_avdtp_free_stream Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_stream(bt_avdtp_stream_t* strm);
```

Description

This is function _bt_avdtp_free_stream.

_bt_avdtp_free_transport_channel Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_transport_channel(bt_avdtp_transport_channel_t* channel);
```

Description

This is function _bt_avdtp_free_transport_channel.

_bt_avdtp_get_control_channel Function

File

avdtp_control.h

C

```
bt_avdtp_control_channel_t* _bt_avdtp_get_control_channel(struct _bt_avdtp_mgr_t* mgr, bt_bdaddr_t*
remote_addr);
```

Description

This is function _bt_avdtp_get_control_channel.

_bt_avdtp_init_cmd_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_cmd_buffers();
```

Description

This is function _bt_avdtp_init_cmd_buffers.

_bt_avdtp_init_sep_config_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_sep_config_buffers();
```

Description

This is function _bt_avdtp_init_sep_config_buffers.

_bt_avdtp_init_signal Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_signal();
```

Description

This is function _bt_avdtp_init_signal.

_bt_avdtp_is_sep_inuse Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_is_sep_inuse(bt_avdtp_mgr_t* mgr, bt_avdtp_sep_t* sep, bt_bdaddr_t* remote_addr);
```

Description

This is function _bt_avdtp_is_sep_inuse.

_bt_avdtp_open_control_channel_ex Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_open_control_channel_ex(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
acl_config);
```

Description

This is function _bt_avdtp_open_control_channel_ex.

_bt_avdtp_open_control_channel_ex Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_open_control_channel_ex(struct _bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
acl_config);
```

Description

This is function _bt_avdtp_open_control_channel_ex.

_bt_avdtp_register_transport_channel_for_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_register_transport_channel_for_operation(bt_avdtp_transport_channel_t* channel);
```

Description

This is function _bt_avdtp_register_transport_channel_for_operation.

_bt_avdtp_send_command Function

File

avdtp_private.h

C

```
void _bt_avdtp_send_command(bt_avdtp_control_channel_t* channel, bt_avdtp_control_cmd_t* cmd);
```

Description

This is function _bt_avdtp_send_command.

_bt_avdtp_send_media_packet Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_send_media_packet(bt_avdtp_stream_t* strm);
```

Description

This is function _bt_avdtp_send_media_packet.

_bt_avdtp_set_signal Function

File

avdtp_private.h

C

```
void _bt_avdtp_set_signal();
```

Description

This is function _bt_avdtp_set_signal.

_bt_avdtp_transport_l2cap_read_data_callback Function

File

avdtp_private.h

C

```
void _bt_avdtp_transport_l2cap_read_data_callback(struct _bt_l2cap_channel_t * pch, bt_byte_p pdata, bt_int
len);
```

Description

This is function _bt_avdtp_transport_l2cap_read_data_callback.

_bt_avdtp_transport_l2cap_state_changed_callback Function

File

avdtp_private.h

C

```
void _bt_avdtp_transport_l2cap_state_changed_callback(bt_l2cap_channel_t* pch, bt_int new_state, void*
param);
```

Description

This is function _bt_avdtp_transport_l2cap_state_changed_callback.

_bt_avdtp_write_caps Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_write_caps(bt_avdtp_mgr_t* mgr, const bt_avdtp_sep_capabilities_t* caps,
bt_avdtp_stream_t* strm, bt_avdtp_control_cmd_t* command);
```

Description

This is function _bt_avdtp_write_caps.

bt_avdtp_abort_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_abort_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Suspend a stream. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_add_media_rx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_add_media_rx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t* buffer);
```

Description

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

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

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

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

bt_avdtp_add_media_tx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_add_media_tx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t* buffer);
```

Description

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

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

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

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

bt_avdtp_cancel_listen Function

File

avdtp.h

C

```
void bt_avdtp_cancel_listen(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte sep_id);
```

Description

brief Cancel listening for incoming connections. ingroup avdtp

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

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

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

bt_avdtp_close_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_close_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Close a stream. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_create_stream Function

File

avdtp.h

C

```
bt_byte bt_avdtp_create_stream(bt_avdtp_mgr_t* mgr);
```

Description

brief Create a stream. ingroup avdtp

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

param mgr AVDTP manager.

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

bt_avdtp_destroy_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_destroy_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Destroy a stream. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_disconnect Function

File

avdtp.h

C

```
bt_bool bt_avdtp_disconnect(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr);
```

Description

brief Disconnect from a remote device. ingroup avdtp

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

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

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

bt_avdtp_discover Function

File

avdtp.h

C

```
bt_bool bt_avdtp_discover(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr);
```

Description

brief Discover SEPs on a remote device. ingroup avdtp

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

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

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

bt_avdtp_find_codec Function

File

avdtp.h

C

```
bt_avdtp_codec_t* bt_avdtp_find_codec(bt_avdtp_mgr_t* mgr, bt_byte codec_type);
```

Description

brief Find a codec ingroup avdtp

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

function returns a pointer to a structure that holds a pointer to a codec's callback function.

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

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

bt_avdtp_get_capabilities Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_capabilities(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte seid_acp);
```

Description

brief Get remote SEP capabilities. ingroup avdtp

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

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

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

bt_avdtp_get_configuration Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_configuration(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream configuration. ingroup avdtp

details This function requests stream configuration from a remote device. As a result of this operation the following events will be generated: @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the p ebt_para.sep_capabilities_received.caps will contain current stream configuration. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the p evt_param.get_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_stream_configuration_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_get_hci_connection Function

File

avdtp.h

C

```
bt_hci_conn_state_t* bt_avdtp_get_hci_connection(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get HCI connection for a stream ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

note This function has not been implemented.

bt_avdtp_get_mgr Function

File

avdtp.h

C

```
bt_avdtp_mgr_t* bt_avdtp_get_mgr();
```

Description

brief Return a pointer to an instance of the AVDTP manager. ingroup avdtp

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

bt_avdtp_get_sep Function

File

avdtp.h

C

```
bt_avdtp_sep_t* bt_avdtp_get_sep(bt_avdtp_mgr_t* mgr, bt_byte sep_id);
```

Description

brief Get a SEP info by its ID. ingroup avdtp

details This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_avdtp_register_sep.

param mgr AVDTP manager. param sep_id The ID of a SEP.

return li c Pointer to bt_avdtp_sep_t if the SEP is in the list of registered SEPs. li c NULL otherwise.

bt_avdtp_get_stream_codec_config Function

File

avdtp.h

C

```
void* bt_avdtp_get_stream_codec_config(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

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

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_avdtp_get_stream_codec_type Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_codec_type(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

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

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

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

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

@arg 0xFF otherwise.

bt_avdtp_get_stream_config Function

File

avdtp.h

C

```
bt_avdtp_sep_capabilities_t* bt_avdtp_get_stream_config(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's configuration. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_avdtp_get_stream_local_sep_id Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_local_sep_id(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's local SEP ID. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_get_stream_remote_address Function

File

avdtp.h

C

```
bt_bdaddr_t* bt_avdtp_get_stream_remote_address(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's remote BT address. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

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

bt_avdtp_get_stream_remote_sep_id Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_remote_sep_id(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's remote SEP ID. ingroup avdtp

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

param mgr AVDTP manager. param strm_handle Stream handle.

return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.

bt_avdtp_get_stream_state Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_state(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get local stream state. ingroup avdtp

details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party.

param mgr AVDTP manager. param strm_handle Stream handle.

return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The stream has been configured. @arg AVDTP_STREAM_STATE_OPEN: The stream has been opened. @arg AVDTP_STREAM_STATE_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets. @arg AVDTP_STREAM_STATE_CLOSING: The stream is closing. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state. @arg AVDTP_STREAM_STATE_ABORTING: The stream is aborting. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

bt_avdtp_init Function

File

avdtp.h

C

```
void bt_avdtp_init();
```

Description

brief Initialize the AVDTP layer. ingroup avdtp

details This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.

bt_avdtp_listen Function

File

avdtp.h

C

```
bt_bool bt_avdtp_listen(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte sep_id);
```

Description

brief Listen for incoming connections. ingroup avdtp

details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_open_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_open_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Open a stream. ingroup avdtp

details This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.open_stream_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the p evt_param.open_stream_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_reconfigure_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_reconfigure_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_avdtp_sep_capabilities_t*
caps);
```

Description

brief Reconfigure stream. ingroup avdtp

details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle. param caps New stream configuration.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_register_callback Function

File

avdtp.h

C

```
void bt_avdtp_register_callback(bt_avdtp_mgr_t* mgr, bt_avdtp_mgr_callback_fp callback, void*
callback_param);
```

Description

brief Register a AVDTP application callback. ingroup avdtp

details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

@arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED:

Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg AVDTP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg AVDTP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after AVDTP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after AVDTP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage). @arg AVDTP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

param mgr AVDTP manager. param callback The callback function that will be called when the AVDTP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_avdtp_register_codec Function

File

avdtp.h

C

```
bt_bool bt_avdtp_register_codec(bt_avdtp_mgr_t* mgr, bt_avdtp_codec_t* codec);
```

Description

brief Register a codec ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function adds a codec's callback function to an internal list.

param mgr AVDTP manager. param codec Pointer to a structure specifying codec type and a callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if there already is a callback for a codec type specified in the p codec parameter.

bt_avdtp_register_sep Function

File

avdtp.h

C

```
bt_byte bt_avdtp_register_sep(bt_avdtp_mgr_t* mgr, bt_byte type, const bt_avdtp_sep_capabilities_t* caps);
```

Description

brief Register a SEP with the local AVDTP manager. ingroup avdtp

details This function is used to make a list of SEPs supported by the local ADVTP entity.

param mgr AVDTP manager. param type The type of a SEP. The type can be one of the following values: @arg AVDTP_SEP_TYPE_SOURCE: The SEP is a source. @arg AVDTP_SEP_TYPE_SINK: The SEP is a sink. param caps The capabilities of a SEP.

return li c ID of a SEP if the function succeeds. li c FALSE otherwise.

bt_avdtp_remove_media_rx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_remove_media_rx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t*
buffer);
```

Description

brief Remove a media packet buffer from a receive queue ingroup avdtp

details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_remove_media_tx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_remove_media_tx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t*
buffer);
```

Description

brief Remove a media packet buffer from a send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media_tx_buffer.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

note This function has not been implemented.

bt_avdtp_security_control Function

File

avdtp.h

C

```
bt_bool bt_avdtp_security_control(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte* sc_data, bt_byte sc_data_len);
```

Description

brief Exchange content protection control data. ingroup avdtp

details This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_STREAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the p evt_param.security_control_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.security_control_completed.err_code == the error code sent by the remote.

note The dotstack does not support content protection. Although the request can be sent it will not affect the operation of the AVDTP in any way.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_set_configuration Function

File

avdtp.h

C

```
bt_bool bt_avdtp_set_configuration(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_bdaddr_t* remote_addr, bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t* caps);
```

Description

brief Set stream configuration. ingroup avdtp

details This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the p evt_param.set_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.set_stream_configuration_completed.err_code == the error code sent by the remote and p evt_param.set_stream_configuration_completed.svc_category == the value of the first Service Category to fail.

param mgr AVDTP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avdtp_start Function

File

avdtp.h

C

```
bt_bool bt_avdtp_start(bt_avdtp_mgr_t* mgr);
```

Description

brief Start the AVDTP layer. ingroup avdtp

details This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

param mgr AVDTP manager.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_start_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_start_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Start a stream. ingroup avdtp

details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_suspend_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_suspend_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Suspend a stream. ingroup avdtp

details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_unregister_codec Function

File

avdtp.h

C

```
bt_bool bt_avdtp_unregister_codec(bt_avdtp_mgr_t* mgr, bt_byte codec_type);
```

Description

brief Unregister a codec ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function removes a codec's callback function from an internal list.

param mgr AVDTP manager. param codec_type Codec type. The codec_type can be one of the following values: @arg AVDTP_CODEC_TYPE_SBC: SBC @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if a callback for a codec type specified in the p codec parameter li has not been previously registered with bt_avdtp_register_codec.

_bt_avdtp_add_param_uintn Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_uintn(bt_avdtp_control_cmd_t* command, bt_uint value);
```

Description

This is function _bt_avdtp_add_param_uintn.

_bt_avdtp_read_caps Function

File

avdtp_private.h

C

```
bt_byte _bt_avdtp_read_caps(bt_avdtp_mgr_t* mgr, bt_avdtp_sep_capabilities_t* caps, bt_byte* data, bt_int
len, bt_int* offset, bt_byte* err_category);
```

Description

This is function _bt_avdtp_read_caps.

bt_avdtp_get_l2cap_channel Function

File

avdtp.h

C

```
bt_l2cap_channel_t* bt_avdtp_get_l2cap_channel(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte
transport_session);
```

Description

This is function bt_avdtp_get_l2cap_channel.

bt_avdtp_clear_media_tx_queue Function

File

avdtp.h

C

```
bt_bool bt_avdtp_clear_media_tx_queue(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Clear send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_avdtp_clear_media_tx_queue.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_get_all_capabilities Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_all_capabilities(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte seid_acp);
```

Description

brief Get remote SEP capabilities. ingroup avdtp

details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following

events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param remote_addr The address of a remote device. param seid_acp The ID of a remote SEP.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_avdtp_get_stream_direction Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_direction(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

This is function bt_avdtp_get_stream_direction.

bt_avdtp_report_delay Function

File

avdtp.h

C

```
bt_bool bt_avdtp_report_delay(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_uint delay);
```

Description

brief Report delay value of a Sink to a Source. ingroup avdtp

details This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the p evt_param.delay_report_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.delay_report_completed.err_code == the error code sent by the Source.

param mgr AVDTP manager. param strm_handle Stream handle. param delay The delay value in 1/10 milliseconds.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_set_media_tx_queue_limit Function

File

avdtp.h

C

```
bt_bool bt_avdtp_set_media_tx_queue_limit(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_uint limit);
```

Description

brief Set limit on the send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of the queue exceeds the set limit the first packet is removed from the queue. The removed packet is not send to the remote device.

param mgr AVDTP manager. param strm_handle Stream handle. param limit Queue limit.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

AVDTP Data Types and Constants

_bt_avdtp_codec_op_param_u Union

File

avdtp.h

C

```
union _bt_avdtp_codec_op_param_u {
  bt_avdtp_codec_op_parse_config_t parse;
  bt_avdtp_codec_op_serialize_config_t serialize;
  bt_avdtp_codec_op_parse_packet_t parse_packet;
};
```

Members

Members	Description
bt_avdtp_codec_op_parse_config_t parse;	< Valid if operation is AVDTP_CODEC_OPCODE_PARSE_CONFIG.
bt_avdtp_codec_op_serialize_config_t serialize;	< Valid if operation is AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG.
bt_avdtp_codec_op_parse_packet_t parse_packet;	< This member is currently not used.

Description

brief Parameter to a codec handler. ingroup avdtp

details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.

_bt_avdtp_sep_t Structure

File

avdtp.h

C

```
struct _bt_avdtp_sep_t {
  bt_avdtp_sep_t* next_sep;
  bt_byte id;
  bt_byte type;
  const bt_avdtp_sep_capabilities_t* caps;
  bt_byte state;
};
```

Members

Members	Description
bt_avdtp_sep_t* next_sep;	< Pointer to next SEP
bt_byte id;	< ID of the SEP
bt_byte type;	< Type of the SEP. This can be one of the following values: < li AVDTP_SEP_TYPE_SOURCE < li AVDTP_SEP_TYPE_SINK
const bt_avdtp_sep_capabilities_t* caps;	< SEP capabilities
bt_byte state;	< State of the SEP buffer. This can be one of the following values: < li AVDTP_SEP_STATE_FREE < li AVDTP_SEP_STATE_IDLE

Description

brief SEP description ingroup avdtp

details This structure is used to hold information about SEPs available on a local device.

_bt_media_packet_t Structure

File

avdtp.h

C

```
struct _bt_media_packet_t {
  bt_media_packet_t* next_packet;
```

```
  bt_byte version;
  bt_byte csrc_count;
  bt_bool marker;
  bt_byte payload_type;
  bt_uint seq_number;
  bt_ulong timestamp;
  bt_ulong ssrc;
  bt_ulong* csrc_list;
  bt_byte* data;
  bt_int data_len;
  bt_int max_data_len;
  bt_cp_header_t cp_header;
};
```

Members

Members	Description
bt_media_packet_t* next_packet;	< Pointer to next buffer
bt_byte version;	< Version of the RTP implementation.
bt_byte csrc_count;	< The CSRC count contains the number of CSRC identifiers that follow the fixed header.
bt_bool marker;	< The interpretation of the marker is defined by a profile. It is < intended to allow significant events such as frame boundaries to < be marked in the packet stream.
bt_byte payload_type;	< This field identifies the format of the RTP payload and determines its interpretation by the application. < A profile specifies default static mapping of payload type codes to payload formats.
bt_uint seq_number;	< The sequence number increments by one for each media packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
bt_ulong timestamp;	< The Time Stamp reflects the sampling instant of the first octet in the media packet.
bt_ulong ssrc;	< The SSRC field identifies the synchronization source. This < identifier is chosen randomly, with the intent that no two synchronization < sources, within the same media transport < session, shall have the same SSRC identifier.
bt_ulong* csrc_list;	< The CSRC list identifies the contributing sources for the payload contained in this packet.
bt_byte* data;	< Pointer to a buffer to store media data. This pointer must be allocated by the AVDTP consumer < before adding this structure to receive (::bt_avdtp_add_media_rx_buffer) or send queue (::bt_avdtp_add_media_tx_buffer).
bt_int data_len;	< Lentgth of the media data.
bt_int max_data_len;	< Maximum length of the media data that can be stored in the buffer pointed by the c data member.
bt_cp_header_t cp_header;	< Content protection header.

Description

brief Media packet buffer ingroup avdtp

details This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of ::bt_avdtp_add_media_rx_buffer and ::bt_avdtp_add_media_tx_buffer.

bt_avdtp_codec_handler_fp Type

File

avdtp.h

C

```
typedef bt_byte (* bt_avdtp_codec_handler_fp)(struct _bt_avdtp_codec_t* codec, bt_byte opcode,
bt_avdtp_codec_op_param_t* op_param, struct _bt_avdtp_mgr_t* mgr);
```

Description

brief Codec handler. ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request.n This typedef defines the interace for the callback function.

param codec A pointer to a structure that describes a codec. param opcode The code of an operation to execute. The c opcode can be one of the following values: @arg AVDTP_CODEC_OPCODE_PARSE_CONFIG: The handler has to parse configuration received from the remote device

and store it in a structure defined by the consumer. @arg AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG: The handler has to serialize the data from a consumer defined structure to a format suitable for sending as a part of a AVDTP request. param op_param A pointer to the operation's specific parameters. Parameters are passed as a pointer to the bt_avdtp_codec_op_param_t union. Which member of the union points to a valid structure depends on the value of the c opcode: @arg bt_avdtp_codec_op_param_t::parse: Valid if c opcode == AVDTP_CODEC_OPCODE_PARSE_CONFIG. @arg bt_avdtp_codec_op_param_t::serialize: Valid if c opcode == AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG. param mgr AVDTP manager.

bt_avdtp_codec_op_decode_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_decode_t {
  void* codec_config;
  bt_byte* in_buffer;
  bt_int in_len;
  bt_byte* out_buffer;
  bt_int out_len;
} bt_avdtp_codec_op_decode_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_encode_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_encode_t {
  void* codec_config;
  bt_byte* in_buffer;
  bt_int in_len;
  bt_byte* out_buffer;
  bt_int out_len;
} bt_avdtp_codec_op_encode_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_param_t Type

File

avdtp.h

C

```
typedef union _bt_avdtp_codec_op_param_u bt_avdtp_codec_op_param_t;
```

Description

This is type bt_avdtp_codec_op_param_t.

bt_avdtp_codec_op_parse_config_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_parse_config_t {
  void* codec_config;
  bt_byte codec_config_max_size;
  bt_byte* buffer;
  bt_int buffer_len;
```

```
  bt_int offset;
} bt_avdtp_codec_op_parse_config_t;
```

Members

Members	Description
void* codec_config;	< A pointer to a structure defined by the AVDTP consumer where codec's configuration < will be stored by the handler. The format of the structure is totally up to < the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC: < @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) < @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) < @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)
bt_byte codec_config_max_size;	< The maximum size of a buffer pointed to by c codec_config field.
bt_byte* buffer;	< A pointer to a buffer holding codec's configuration in OTA format.
bt_int buffer_len;	< The length of the c buffer.
bt_int offset;	< The c buffer points to a complete packet received from the remoted device. The c offset < points to a location in the c buffer where codec's configuration starts.

Description

brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp

details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.

bt_avdtp_codec_op_parse_packet_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_parse_packet_t {
  void* packet_info;
  bt_media_packet_t* packet;
} bt_avdtp_codec_op_parse_packet_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_serialize_config_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_serialize_config_t {
  void* codec_config;
  bt_byte* buffer;
  bt_int buffer_len;
  bt_int offset;
} bt_avdtp_codec_op_serialize_config_t;
```

Members

Members	Description
void* codec_config;	< A pointer to a structure defined by the AVDTP consumer where codec's configuration < will be read from by the handler. The format of the structure is totally up to < the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC: < @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) < @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) < @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)
bt_byte* buffer;	< A pointer to a buffer where the handler has to write codec's configuration in OTA format.
bt_int buffer_len;	< The length of the c buffer.
bt_int offset;	< The c buffer points to a complete packet that will be sent to the remote device. The c offset < points to a location in the c buffer where codec's configuration has to be written.

Description

brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp

details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.

bt_avdtp_codec_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_t {
  struct _bt_avdtp_codec_t* next_codec;
  bt_byte codec_type;
  bt_avdtp_codec_handler_fp codec_handler;
} bt_avdtp_codec_t;
```

Members

Members	Description
bt_byte codec_type;	< Codec type. < The c codec_type can be one of the following values: < @arg AVDTP_CODEC_TYPE_SBC: SBC < @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) < @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) < @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) < @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec
bt_avdtp_codec_handler_fp codec_handler;	< A pointer to a codec handler.

Description

brief Codec handler description. ingroup avdtp

details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.

bt_avdtp_control_channel_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_control_channel_t {
  bt_byte state;
  bt_bdaddr_t remote_addr;
  bt_l2cap_channel_t* l2cap_channel;
  bt_byte* tx_buffer;
  bt_queue_element_t* send_cq_head;
  bt_queue_element_t* ack_cq_head;
  bt_byte next_transaction_id;
  bt_byte next_ts_id;
  bt_byte next_tc_id;
  struct _bt_avdtp_mgr_t* avdtp_mgr;
  bt_byte remaining_connect_attemtps;
  bt_bdaddr_t connect_address;
  bt_uint connect_acl_config;
} bt_avdtp_control_channel_t;
```

Description

This is type bt_avdtp_control_channel_t.

bt_avdtp_control_cmd_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_control_cmd_t {
  struct _bt_avdtp_control_cmd_t* next_cmd;
  bt_byte code;
  bt_byte type;
  bt_byte transaction_id;
  bt_byte* params;
  bt_uint params_len;
} bt_avdtp_control_cmd_t;
```

Description

This is type bt_avdtp_control_cmd_t.

bt_avdtp_ctrl_evt_data_received_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_ctrl_evt_data_received_t {
  bt_byte* data;
  bt_int len;
} bt_avdtp_ctrl_evt_data_received_t;
```

Description

This is type bt_avdtp_ctrl_evt_data_received_t.

bt_avdtp_event_t Union

File

avdtp.h

C

```
typedef union _bt_avdtp_event_u {
  bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;
  bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;
  bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;
  bt_avdtp_evt_discover_completed_t discover_completed;
  bt_avdtp_evt_sep_info_received_t sep_info_received;
  bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;
  bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;
  bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;
  bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;
  bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;
  bt_avdtp_evt_open_stream_completed_t open_stream_completed;
  bt_avdtp_evt_start_stream_completed_t start_stream_completed;
  bt_avdtp_evt_close_stream_completed_t close_stream_completed;
  bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;
  bt_avdtp_evt_stream_security_control_completed_t security_control_completed;
  bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;
  bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;
  bt_avdtp_evt_open_stream_requested_t open_stream_requested;
  bt_avdtp_evt_start_stream_requested_t start_stream_requested;
  bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;
  bt_avdtp_evt_close_stream_requested_t close_stream_requested;
  bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;
  bt_avdtp_evt_delay_report_completed_t delay_report_completed;
  bt_avdtp_evt_set_stream_configuration_t set_stream_configuration;
  bt_avdtp_evt_stream_configured_t stream_configured;
  bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;
  bt_avdtp_evt_stream_opened_t stream_opened;
  bt_avdtp_evt_stream_started_t stream_started;
  bt_avdtp_evt_stream_suspended_t stream_suspended;
  bt_avdtp_evt_stream_closed_t stream_closed;
  bt_avdtp_evt_stream_aborted_t stream_aborted;
  bt_avdtp_evt_media_packet_received_t media_packet_received;
  bt_avdtp_evt_media_packet_sent_t media_packet_sent;
```

```
    bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;
  } bt_avdtp_event_t;
```

Members

Members	Description
bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;	< Valid if event is AVDTP_EVT_CTRL_CHANNEL_CONNECTED
bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;	< Valid if event is AVDTP_EVT_CTRL_CONNECTION_FAILED
bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;	< Valid if event is AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED
bt_avdtp_evt_discover_completed_t discover_completed;	< Valid if event is AVDTP_EVT_DISCOVER_COMPLETED
bt_avdtp_evt_sep_info_received_t sep_info_received;	< Valid if event is AVDTP_EVT_SEP_INFO_RECEIVED
bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;	< Valid if event is AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED
bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;	< Valid if event is AVDTP_EVT_SEP_CAPABILITIES_RECEIVED
bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;	< Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;	< Valid if event is AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;	< Valid if event is AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED
bt_avdtp_evt_open_stream_completed_t open_stream_completed;	< Valid if event is AVDTP_EVT_OPEN_STREAM_COMPLETED
bt_avdtp_evt_start_stream_completed_t start_stream_completed;	< Valid if event is AVDTP_EVT_START_STREAM_COMPLETED
bt_avdtp_evt_close_stream_completed_t close_stream_completed;	< Valid if event is AVDTP_EVT_CLOSE_STREAM_COMPLETED
bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;	< Valid if event is AVDTP_EVT_SUSPEND_STREAM_COMPLETED
bt_avdtp_evt_stream_security_control_completed_t security_control_completed;	< Valid if event is AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED
bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;	< Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED
bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;	< Valid if event is AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED
bt_avdtp_evt_open_stream_requested_t open_stream_requested;	< Valid if event is AVDTP_EVT_OPEN_STREAM_REQUESTED
bt_avdtp_evt_start_stream_requested_t start_stream_requested;	< Valid if event is AVDTP_EVT_START_STREAM_REQUESTED
bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;	< Valid if event is AVDTP_EVT_SUSPEND_STREAM_REQUESTED
bt_avdtp_evt_close_stream_requested_t close_stream_requested;	< Valid if event is AVDTP_EVT_CLOSE_STREAM_REQUESTED
bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;	< Valid if event is AVDTP_EVT_ABORT_STREAM_REQUESTED
bt_avdtp_evt_delay_report_completed_t delay_report_completed;	< Valid if event is AVDTP_EVT_DELAYREPORT_COMPLETED
bt_avdtp_evt_stream_configured_t stream_configured;	< Valid if event is AVDTP_EVT_STREAM_CONFIGURED
bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;	< Valid if event is AVDTP_EVT_STREAM_RECONFIGURED
bt_avdtp_evt_stream_opened_t stream_opened;	< Valid if event is AVDTP_EVT_STREAM_OPENED
bt_avdtp_evt_stream_started_t stream_started;	< Valid if event is AVDTP_EVT_STREAM_STARTED
bt_avdtp_evt_stream_suspended_t stream_suspended;	< Valid if event is AVDTP_EVT_STREAM_SUSPENDED
bt_avdtp_evt_stream_closed_t stream_closed;	< Valid if event is AVDTP_EVT_STREAM_CLOSED
bt_avdtp_evt_stream_aborted_t stream_aborted;	< Valid if event is AVDTP_EVT_STREAM_ABORTED
bt_avdtp_evt_media_packet_received_t media_packet_received;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_RECEIVED

bt_avdtp_evt_media_packet_sent_t media_packet_sent;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_SENT
bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_SEND_FAILED

Description

brief Parameter to an application callback. ingroup avdtp

details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

bt_avdtp_evt_abort_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_abort_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_abort_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to abort.

Description

brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.

bt_avdtp_evt_close_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_close_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_close_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.

bt_avdtp_evt_close_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_close_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_close_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to close.

Description

brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.

bt_avdtp_evt_ctrl_channel_connected_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_channel_connected_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_channel_connected_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.

bt_avdtp_evt_ctrl_channel_disconnected_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_channel_disconnected_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_channel_disconnected_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.

bt_avdtp_evt_discover_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_discover_completed_t {
  bt_byte err_code;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_discover_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of discovering. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.

bt_avdtp_evt_get_sep_capabilities_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_get_sep_capabilities_completed_t {
  bt_byte err_code;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_get_sep_capabilities_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.

bt_avdtp_evt_get_stream_configuration_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_get_stream_configuration_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_get_stream_configuration_completed_t;
```

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.

bt_avdtp_evt_media_packet_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_received_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_received_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.

bt_avdtp_evt_media_packet_send_failed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_send_failed_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_send_failed_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.

bt_avdtp_evt_media_packet_sent_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_sent_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_sent_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.

bt_avdtp_evt_open_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_open_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_open_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.

bt_avdtp_evt_open_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_open_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_open_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to open.

Description

brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.

bt_avdtp_evt_reconfigure_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_reconfigure_stream_requested_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
} bt_avdtp_evt_reconfigure_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte err_category;	< If local device cannot accept the request it should set c err_category to the value of the forst Service Category that failed.
bt_avdtp_sep_t* sep;	< Description of the local SEP.
bt_avdtp_sep_capabilities_t* config;	< Stream configuration requested by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.

bt_avdtp_evt_sep_capabilities_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_sep_capabilities_received_t {
  bt_avdtp_sep_capabilities_t* caps;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_sep_capabilities_received_t;
```

Members

Members	Description
bt_avdtp_sep_capabilities_t* caps;	< SEP capabilities or stream configuration.

Description

brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.

bt_avdtp_evt_sep_info_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_sep_info_received_t {
  bt_byte sep_id;
```

```
  bt_byte sep_type;
  bt_bool in_use;
  bt_byte media_type;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_sep_info_received_t;
```

Members

Members	Description
bt_byte sep_id;	< SEP ID.
bt_byte sep_type;	< SEP type. This can be either AVDTP_SEP_TYPE_SOURCE or AVDTP_SEP_TYPE_SINK.
bt_bool in_use;	< A flag indicating if a SEP is already being used.
bt_byte media_type;	< Type of media supported by this SEP. This can be on of the following values: < li AVDTP_MEDIA_TYPE_AUDIO < li AVDTP_MEDIA_TYPE_VIDEO < li AVDTP_MEDIA_TYPE_MULTIMEDIA

Description

brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.

bt_avdtp_evt_set_stream_configuration_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
  bt_byte svc_category;
} bt_avdtp_evt_set_stream_configuration_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.
bt_byte svc_category;	< The value of the first Service Category to fail if the remote rejected the request.

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.

bt_avdtp_evt_set_stream_configuration_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_requested_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_byte int_sep_id;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
} bt_avdtp_evt_set_stream_configuration_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte err_category;	< If local device cannot accept the request it should set c err_category to the value of the forst Service Category that failed.
bt_avdtp_sep_t* sep;	< Description of the local SEP.
bt_byte int_sep_id;	< The ID of the remote SEP.
bt_avdtp_sep_capabilities_t* config;	< Stream configuration requested by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.

bt_avdtp_evt_start_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_start_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_start_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

bt_avdtp_evt_start_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_start_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_start_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to start.

Description

brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.

bt_avdtp_evt_stream_aborted_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_aborted_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_aborted_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been aborted.

Description

brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

bt_avdtp_evt_stream_closed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_closed_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_closed_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been closed.

Description

brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.

bt_avdtp_evt_stream_configured_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_configured_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_configured_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been configured.

Description

brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.

bt_avdtp_evt_stream_opened_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_opened_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_opened_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been opened.

Description

brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.

bt_avdtp_evt_stream_reconfigure_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_reconfigure_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
  bt_byte svc_category;
} bt_avdtp_evt_stream_reconfigure_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.
bt_byte svc_category;	< The value of the first Service Category to fail if the remote rejected the request.

Description

brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.

bt_avdtp_evt_stream_reconfigured_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_reconfigured_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_reconfigured_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been reconfigured.

Description

brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.

bt_avdtp_evt_stream_security_control_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_security_control_completed_t {
  bt_byte err_code;
} bt_avdtp_evt_stream_security_control_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.

Description

brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.

bt_avdtp_evt_stream_started_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_started_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_started_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been started.

Description

brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.

bt_avdtp_evt_stream_suspended_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_suspended_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_suspended_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been suspended.

Description

brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

bt_avdtp_evt_suspend_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_suspend_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_suspend_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.

bt_avdtp_evt_suspend_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_suspend_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_suspend_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to suspend.

Description

brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.

bt_avdtp_mgr_callback_fp Type

File

avdtp.h

C

```
typedef void (* bt_avdtp_mgr_callback_fp)(struct _bt_avdtp_mgr_t* mgr, bt_byte evt, bt_avdtp_event_t*
evt_param, void* callback_param);
```

Description

brief AVDTP application callback. ingroup avdtp

details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

param mgr AVDTP manager.

param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg AVDTP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg AVDTP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after AVDTP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after AVDTP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage). @arg AVDTP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

param evt_param Event parameter. Which member of the bt_avdtp_event_t union is valid depends on the event: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: c bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: c bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: c NULL @arg AVDTP_EVT_DISCOVER_COMPLETED: c bt_avdtp_evt_discover_completed_t discover_completed; @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: c bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: c bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: c bt_avdtp_evt_open_stream_completed_t open_stream_completed @arg AVDTP_EVT_START_STREAM_COMPLETED: c bt_avdtp_evt_start_stream_completed_t start_stream_completed @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: c bt_avdtp_evt_close_stream_completed_t close_stream_completed @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: c bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: c bt_avdtp_evt_stream_security_control_completed_t security_control_completed @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg AVDTP_EVT_SEP_INFO_RECEIVED: c bt_avdtp_evt_sep_info_received_t sep_info_received @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: c bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: c bt_avdtp_evt_open_stream_requested_t open_stream_requested @arg AVDTP_EVT_START_STREAM_REQUESTED: c bt_avdtp_evt_start_stream_requested_t start_stream_requested @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: c bt_avdtp_evt_close_stream_requested_t close_stream_requested @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: c bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested @arg

AVDTP_EVT_ABORT_STREAM_REQUESTED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: c bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: c bt_avdtp_evt_media_packet_received_t media_packet_received @arg AVDTP_EVT_STREAM_CONFIGURED: c bt_avdtp_evt_stream_configured_t stream_configured @arg AVDTP_EVT_STREAM_RECONFIGURED: c bt_avdtp_evt_stream_reconfigured_t stream_reconfigured @arg AVDTP_EVT_STREAM_OPENED: c bt_avdtp_evt_stream_opened_t stream_opened @arg AVDTP_EVT_STREAM_STARTED: c bt_avdtp_evt_stream_started_t stream_started @arg AVDTP_EVT_STREAM_CLOSED: c bt_avdtp_evt_stream_closed_t stream_closed @arg AVDTP_EVT_STREAM_SUSPENDED: c bt_avdtp_evt_stream_suspended_t stream_suspended @arg AVDTP_EVT_STREAM_ABORTED: c bt_avdtp_evt_stream_aborted_t stream_aborted @arg AVDTP_EVT_MEDIA_PACKET_SENT: c bt_avdtp_evt_media_packet_sent_t media_packet_sent @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: c bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed

param callback_param A pointer to an arbitrary data set by a call to bt_avdtp_register_callback.

bt_avdtp_mgr_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_mgr_t {
  bt_byte state;
  bt_byte flags;
  bt_avdtp_sep_t* seps;
  bt_byte next_sep_id;
  bt_avdtp_stream_t* streams;
  bt_byte next_stream_handle;
  bt_avdtp_control_channel_t* control_channels;
  bt_avdtp_transport_channel_t* transport_channels;
  bt_avdtp_codec_t* codecs;
  bt_avdtp_mgr_callback_fp callback;
  void* callback_param;
  bt_avdtp_control_cmd_t* pending_command;
  bt_avdtp_stream_t* opening_strm;
  bt_avdtp_stream_t* sending_strm;
} bt_avdtp_mgr_t;
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li AVDTP_MANAGER_STATE_IDLE < li AVDTP_MANAGER_STATE_CONNECTING
bt_byte flags;	< Additional manager state. This value can be a combination of the following values: < li AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET
bt_avdtp_sep_t* seps;	< Pointer to a buffer of SEPs available for allocating with c bt_avdtp_register_sep.
bt_byte next_sep_id;	< Holds ID of the next SEP to be allocated. Every time c bt_avdtp_register_sep is called this value is increased by 1.
bt_avdtp_stream_t* streams;	< Pointer to a buffer of streams available for allocating with c bt_avdtp_create_stream.
bt_byte next_stream_handle;	< Holds ID of the next stream to be allocated. Every time c bt_avdtp_create_stream is called this value is increased by 1.
bt_avdtp_control_channel_t* control_channels;	< Pointer to a buffer of available control channles.
bt_avdtp_transport_channel_t* transport_channels;	< Pointer to a buffer of available transport channles.
bt_avdtp_codec_t* codecs;	< A list of supported codecs.
bt_avdtp_mgr_callback_fp callback;	< Pointer to a function which a AVDTP consumer must register in order to be notified of various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.
bt_avdtp_control_cmd_t* pending_command;	< If local device wants to send a command to a remote device but control channel does not exists this member holds a pointer to the command until the channel is created.
bt_avdtp_stream_t* opening_strm;	< Holds a pointer to a stream being opened by a remote device.
bt_avdtp_stream_t* sending_strm;	< Holds a pointer to a stream sending media packet.

Description

brief AVDTP manager. ingroup avdtp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().

bt_avdtp_sep_capabilities_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_sep_capabilities_t {
  bt_byte categories;
  bt_byte media_type;
  bt_byte codec_type;
  void* codec_config;
  bt_uint cp_type;
  bt_byte* cp_specific;
  bt_byte cp_specific_len;
  bt_byte recovery_type;
  bt_byte max_recovery_window;
  bt_byte max_parity_code_packets;
  bt_byte header_compression_options;
  bt_byte multiplexing_options;
  bt_byte tsid_media;
  bt_byte tcid_media;
  bt_byte tsid_reporting;
  bt_byte tcid_reporting;
  bt_byte tsid_recovery;
  bt_byte tcid_recovery;
} bt_avdtp_sep_capabilities_t;
```

Members

Members	Description
bt_byte categories;	< Defines service capabilities exposed by a SEP to a remote party. This can be a combination of the following values: < li AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT < li AVDTP_SEP_CAPABILITY_FLAG_REPORTING < li AVDTP_SEP_CAPABILITY_FLAG_RECOVERY < li AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION < li AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION < li AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING < li AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC
bt_byte media_type;	< Type of media supported by this SEP. This can be on of the following values: < li AVDTP_MEDIA_TYPE_AUDIO < li AVDTP_MEDIA_TYPE_VIDEO < li AVDTP_MEDIA_TYPE_MULTIMEDIA
bt_byte codec_type;	< Codec type supported by this SEP. This can be on of the following values: < li AVDTP_CODEC_TYPE_SBC < li AVDTP_CODEC_TYPE_MPEG1_2_AUDIO < li AVDTP_CODEC_TYPE_MPEG2_4_AAC < li AVDTP_CODEC_TYPE_ATRAC < li AVDTP_CODEC_TYPE_NON_A2DP
void* codec_config;	< Pointer to a buffer that holds codec specific capabilities
bt_uint cp_type;	< Type of content protection supported by this SEP. < note Content protection is currently not supported by dotstack.
bt_byte* cp_specific;	< Pointer to a buffer holding content protection specific data. < note Content protection is currently not supported by dotstack.
bt_byte cp_specific_len;	< Length of the content protection specific data. < note Content protection is currently not supported by dotstack.
bt_byte recovery_type;	< Type of recovery supported by this SEP. < note Recovery is currently not supported by dotstack.
bt_byte max_recovery_window;	< Recovery window size. < note Recovery is currently not supported by dotstack.
bt_byte max_parity_code_packets;	< Maximum number of parity codec packets. < note Recovery is currently not supported by dotstack.
bt_byte header_compression_options;	< Header compression configuration < note Header compression is currently not supported by dotstack.
bt_byte multiplexing_options;	< Multiplexing configuration < note Multiplexing is currently not supported by dotstack.
bt_byte tsid_media;	< ID of the media transport session
bt_byte tcid_media;	< ID of the media transport channel
bt_byte tsid_reporting;	< ID of the reporting transport session
bt_byte tcid_reporting;	< ID of the reporting transport channel
bt_byte tsid_recovery;	< ID of the recovery transport session

bt_byte tcid_recovery;	< ID of the recovery transport channel

Description

brief SEP capabilities ingroup avdtp

details This structure is used to hold SEP capabilities.

bt_avdtp_sep_t Type

File

avdtp.h

C

```
typedef struct _bt_avdtp_sep_t bt_avdtp_sep_t;
```

Description

This is type bt_avdtp_sep_t.

bt_avdtp_stream_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_stream_t {
  struct _bt_avdtp_stream_t* next_stream;
  bt_byte handle;
  bt_byte state;
  bt_byte flags;
  struct _bt_avdtp_sep_t* sep;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte remote_seid;
  bt_bdaddr_t remote_addr;
  bt_avdtp_transport_session_t sessions[AVDTP_MAX_STREAM_TRANSPORT_SESSION];
  struct _bt_avdtp_mgr_t* mgr;
  bt_byte cur_channel_index;
  bt_byte* listen_sep_list;
  bt_byte listen_sep_count;
  bt_queue_element_t* media_rx_queue;
  bt_queue_element_t* media_tx_queue;
  bt_uint media_tx_queue_limit;
} bt_avdtp_stream_t;
```

Members

Members	Description
struct _bt_avdtp_stream_t* next_stream;	< Pointer to next stream.
bt_byte handle;	< Stream handle. This values is allocated by dotstack and is used to manipulate the stream by the AVDTP consumer.
bt_byte state;	< State of the stream. This value can be one of the following values: < li AVDTP_STREAM_STATE_IDLE < li AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS < li AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS < li AVDTP_STREAM_STATE_CONFIGURED < li AVDTP_STREAM_STATE_OPEN < li AVDTP_STREAM_STATE_STREAMING < li AVDTP_STREAM_STATE_CLOSING < li AVDTP_STREAM_STATE_ABORTING
bt_byte flags;	< Additional stream state. This value can be one of the following values: < li AVDTP_STREAM_FLAG_LISTENING
struct _bt_avdtp_sep_t* sep;	< Local SEP this stream is connected to.
bt_avdtp_sep_capabilities_t* config;	< Current SEP configuration.
bt_byte remote_seid;	< ID of the remote SEP.
bt_bdaddr_t remote_addr;	< BT address of the remote device.
bt_avdtp_transport_session_t sessions[AVDTP_MAX_STREAM_TRANSPORT_SESSION];	< List of transport session available/active on this stream. < note There can be up to 3 (AVDTP_MAX_STREAM_TRANSPORT_SESSION == 3) transport sessions on a stream - media, reporting and recovery. < dotstack supports only media sessions so the other two are never used.

struct _bt_avdtp_mgr_t* mgr;	< AVDTP manager this stream belongs to.
bt_byte cur_channel_index;	< This value is currenlty not used.
bt_byte* listen_sep_list;	< A list of SEPs this channel this channel is listening on (i.e. can accept incoming connection requests).
bt_byte listen_sep_count;	< The number of SEPs in c listen_sep_list.
bt_queue_element_t* media_rx_queue;	< A list of media packet buffer for receiving incoming packets.
bt_queue_element_t* media_tx_queue;	< A list of media packet buffer to be sent to a remote device.
bt_uint media_tx_queue_limit;	< Maximum number of packets that TX queue can hold. 0 - unlimited.

Description

brief Stream description ingroup avdtp

details This structure is used to hold information about streams available on a local device.

bt_avdtp_transport_channel_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_transport_channel_t {
  struct _bt_avdtp_transport_channel_t* next_channel;
  bt_byte id;
  bt_byte type;
  bt_l2cap_channel_t* l2cap_channel;
  bt_byte ref_count;
  bt_byte connect_ref_count;
} bt_avdtp_transport_channel_t;
```

Members

Members	Description
struct _bt_avdtp_transport_channel_t* next_channel;	< Pointer to next channel.
bt_byte id;	< ID of the channel.
bt_byte type;	< Type of the channel. This can be one of the following values: < li AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED < li AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED < note Shared channles (i.e. multiplexing) is currently not supported by dotstack
bt_l2cap_channel_t* l2cap_channel;	< L2CAp channel used to transfer this AVDTP channel's data.
bt_byte ref_count;	< Channel's reference count. This value is intended for use with shared channels. < When ref count reaches 0 the channel is closed. < note Shared channles (i.e. multiplexing) is currently not supported by dotstack
bt_byte connect_ref_count;	< This value is currently not used.

Description

brief Transport channel description ingroup avdtp

details This structure is used to hold information about transport channels available on a local device.

bt_avdtp_transport_op_callback_fp Type

File

avdtp_private.h

C

```
typedef void (* bt_avdtp_transport_op_callback_fp)(bt_byte status, bt_byte opcode, void* param);
```

Description

This is type bt_avdtp_transport_op_callback_fp.

bt_avdtp_transport_session_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_transport_session_t {
  bt_byte id;
  bt_byte type;
  bt_avdtp_transport_channel_t* transport_channel;
} bt_avdtp_transport_session_t;
```

Members

Members	Description
bt_byte id;	< ID of the transport session.
bt_byte type;	< Type of the transport session. This can be one of the following values: < li AVDTP_TRANSPORT_SESSION_TYPE_MEDIA < li AVDTP_TRANSPORT_SESSION_TYPE_REPORTING < li AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY
bt_avdtp_transport_channel_t* transport_channel;	< Transport channel used for this transport session.

Description

brief Transport session description ingroup avdtp

details This structure is used to hold information about transport sessions available on a local device.

bt_media_packet_t Type

File

avdtp.h

C

```
typedef struct _bt_media_packet_t bt_media_packet_t;
```

Description

This is type bt_media_packet_t.

__AVDTP_CONFIG_H Macro

File

avdtp_config.h

C

```
#define __AVDTP_CONFIG_H
```

Description

This is macro __AVDTP_CONFIG_H.

__AVDTP_CONTROL_H Macro

File

avdtp_control.h

C

```
#define __AVDTP_CONTROL_H
```

Description

This is macro __AVDTP_CONTROL_H.

__AVDTP_H Macro

File

avdtp.h

C

```
#define __AVDTP_H
```

Description

This is macro __AVDTP_H.

__AVDTP_PRIVATE_H Macro

File

avdtp_private.h

C

```
#define __AVDTP_PRIVATE_H
```

Description

This is macro __AVDTP_PRIVATE_H.

AVDTP_CODEC_OPCODE_PARSE_CONFIG Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_PARSE_CONFIG 0          ///< Parse codec configuration.
```

Description

< Parse codec configuration.

AVDTP_CODEC_OPCODE_PARSE_PACKET Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_PARSE_PACKET 2
```

Description

This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.

AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG 1          ///< Serialize codec configuration.
```

Description

< Serialize codec configuration.

AVDTP_CODEC_TYPE_ATRAC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_ATRAC 0x04 ///< ATRAC (proprietary codec owned by Sony Corporation).
```

Description

< ATRAC (proprietary codec owned by Sony Corporation).

AVDTP_CODEC_TYPE_MPEG1_2_AUDIO Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_MPEG1_2_AUDIO 0x01 ///< MPEG-1,2 (optional).
```

Description

< MPEG-1,2 (optional).

AVDTP_CODEC_TYPE_MPEG2_4_AAC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_MPEG2_4_AAC 0x02 ///< MPEG-2,4 AAC (optional, used in Apple's products).
```

Description

< MPEG-2,4 AAC (optional, used in Apple's products).

AVDTP_CODEC_TYPE_NON_A2DP Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_NON_A2DP 0xFF ///< Vendor specific.
```

Description

< Vendor specific.

AVDTP_CODEC_TYPE_SBC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_SBC 0x00 ///< SBC (mandatory to support in A2DP profile).
```

Description

< SBC (mandatory to support in A2DP profile).

AVDTP_CTLR_CHANNEL_EVT_CONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_CONNECTED 1
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.

AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED 3
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.

AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED 2
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.

AVDTP_CTLR_CHANNEL_EVT_NOTHING Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_NOTHING 0
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.

AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED 4
```

Description

This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.

AVDTP_CTRL_CHANNEL_STATE_CONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_CONNECTED 1
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.

AVDTP_CTRL_CHANNEL_STATE_CONNECTING Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.

AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED 0
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.

AVDTP_CTRL_MESSAGE_TYPE_ACCEPT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_ACCEPT 2
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.

AVDTP_CTRL_MESSAGE_TYPE_COMMAND Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_COMMAND 0
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.

AVDTP_CTRL_MESSAGE_TYPE_FLD Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_FLD 3
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.

AVDTP_CTRL_MESSAGE_TYPE_REJECT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_REJECT 3
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.

AVDTP_CTRL_PACKET_TYPE_CONTINUE Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_CONTINUE 0x08
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.

AVDTP_CTRL_PACKET_TYPE_END Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_END 0x0c
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_END.

AVDTP_CTRL_PACKET_TYPE_FLD Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_FLD 0x0c
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_FLD.

AVDTP_CTRL_PACKET_TYPE_SIGNLE Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_SIGNLE 0
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.

AVDTP_CTRL_PACKET_TYPE_START Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_START 0x04
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_START.

AVDTP_ERROR_BAD_ACP_SEID Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_ACP_SEID 0x12    ///< The requested command indicates an invalid ACP SEP ID (not addressable)
```

Description

< The requested command indicates an invalid ACP SEP ID (not addressable)

AVDTP_ERROR_BAD_CP_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_CP_FORMAT 0x27    ///< The format of Content Protection Service Capability is not correct.
```

Description

< The format of Content Protection Service Capability is not correct.

AVDTP_ERROR_BAD_HEADER_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_HEADER_FORMAT 0x01    ///< The request packet header format is invalid.
```

Description

< The request packet header format is invalid.

AVDTP_ERROR_BAD_LENGTH Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_LENGTH 0x11    ///< The request packet length is not match the assumed length.
```

Description

< The request packet length is not match the assumed length.

AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT 0x23    ///< The format of Media Transport Capability is not
correct.
```

Description

< The format of Media Transport Capability is not correct.

AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT 0x28    ///< The format of Multiplexing Service Capability is
not correct.
```

Description

< The format of Multiplexing Service Capability is not correct.

AVDTP_ERROR_BAD_PAYLOAD_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_PAYLOAD_FORMAT 0x18    ///< The requested command has an incorrect payload format.
```

Description

< The requested command has an incorrect payload format.

AVDTP_ERROR_BAD_RECOVERY_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_RECOVERY_FORMAT 0x25    ///< The format of Recovery Service Capability is not
correct.
```

Description

< The format of Recovery Service Capability is not correct.

AVDTP_ERROR_BAD_RECOVERY_TYPE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_RECOVERY_TYPE 0x22    ///< The requested Recovery Type is not defined in AVDTP.
```

Description

< The requested Recovery Type is not defined in AVDTP.

AVDTP_ERROR_BAD_ROHC_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_ROHC_FORMAT 0x26    ///< The format of Header Compression Service Capability is not
correct.
```

Description

< The format of Header Compression Service Capability is not correct.

AVDTP_ERROR_BAD_SERV_CATEGORY Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_SERV_CATEGORY 0x17    ///< The value of Service Category in the request packet is
not defined in AVDTP.
```

Description

< The value of Service Category in the request packet is not defined in AVDTP.

AVDTP_ERROR_BAD_STATE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_STATE 0x31    ///< The stream is in state that does not permit executing commands.
```

Description

< The stream is in state that does not permit executing commands.

AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL 0x41    ///< An attempt to establish a control channel has
failed.
```

Description

< An attempt to establish a control channel has failed.

AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT 0x40    ///< An attempt to establish a transport channel
has failed.
```

Description

< An attempt to establish a transport channel has failed.

AVDTP_ERROR_INVALID_CAPABILITIES Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_INVALID_CAPABILITIES 0x1a    ///< The reconfigure command is an attempt to reconfigure
a transport service capabilities of the SEP. Reconfigure is only permitted for application service
capabilities.
```

Description

< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.

AVDTP_ERROR_NOT_SUPPORTED_COMMAND Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_NOT_SUPPORTED_COMMAND 0x19    ///< The requested command is not supported by the device.
```

Description

< The requested command is not supported by the device.

AVDTP_ERROR_SEP_IN_USE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SEP_IN_USE 0x13    ///< The SEP is in use.
```

Description

< The SEP is in use.

AVDTP_ERROR_SEP_NOT_IN_USE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SEP_NOT_IN_USE 0x14    ///< The SEP is not in use.
```

Description

< The SEP is not in use.

AVDTP_ERROR_SUCCESS Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SUCCESS 0       ///< The operation completed with no errors.
```

Description

< The operation completed with no errors.

AVDTP_ERROR_UNSUPPORTED_CONFIGURAION Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_UNSUPPORTED_CONFIGURAION 0x29    ///< Configuration not supported.
```

Description

< Configuration not supported.

AVDTP_EVT_ABORT_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_ABORT_STREAM_COMPLETED 14    ///< This event is generated when a local device received a
response (either positive or negative) to a "abort stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.

AVDTP_EVT_ABORT_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_ABORT_STREAM_REQUESTED 55    ///< This event is generated when a local device received
"abort stream" request.
```

Description

< This event is generated when a local device received "abort stream" request.

AVDTP_EVT_CLOSE_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CLOSE_STREAM_COMPLETED 11    ///< This event is generated when a local device received a
response (either positive or negative) to a "close stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.

AVDTP_EVT_CLOSE_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CLOSE_STREAM_REQUESTED 53    ///< This event is generated when a local device received "close stream" request.
```

Description

< This event is generated when a local device received "close stream" request.

AVDTP_EVT_CTRL_CHANNEL_CONNECTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CHANNEL_CONNECTED 1     ///< This event is generated when a control channel between two AVDTP entities has been established.
```

Description

< This event is generated when a control channel between two AVDTP entities has been established.

AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED 2     ///< This event is generated when a control channel between two AVDTP entities has been terminated.
```

Description

< This event is generated when a control channel between two AVDTP entities has been terminated.

AVDTP_EVT_CTRL_CONNECTION_FAILED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CONNECTION_FAILED 3     ///< This event is generated when a local device failed to create a control channel between two AVDTP entities.
```

Description

< This event is generated when a local device failed to create a control channel between two AVDTP entities.

AVDTP_EVT_DISCOVER_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_DISCOVER_COMPLETED 4     ///< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "discover" request.

AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED 5     ///< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.

AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED 7     ///< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.

AVDTP_EVT_LAST Macro

File

avdtp.h

C

```
#define AVDTP_EVT_LAST 99
```

Description

This is macro AVDTP_EVT_LAST.

AVDTP_EVT_MEDIA_PACKET_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_RECEIVED 57    ///< This event is generated when a local device received a media packet.
```

Description

< This event is generated when a local device received a media packet.

AVDTP_EVT_MEDIA_PACKET_SEND_FAILED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_SEND_FAILED 66    ///< This event is generated when a local device failed to send a media packet.
```

Description

< This event is generated when a local device failed to send a media packet.

AVDTP_EVT_MEDIA_PACKET_SENT Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_SENT 65    ///< This event is generated when a local device sent a media packet.
```

Description

< This event is generated when a local device sent a media packet.

AVDTP_EVT_NULL Macro

File

avdtp.h

C

```
#define AVDTP_EVT_NULL 0
```

Description

This is macro AVDTP_EVT_NULL.

AVDTP_EVT_OPEN_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_OPEN_STREAM_COMPLETED 9     ///< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.

AVDTP_EVT_OPEN_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_OPEN_STREAM_REQUESTED 51    ///< This event is generated when a local device received "open stream" request.
```

Description

< This event is generated when a local device received "open stream" request.

AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED 56    ///< This event is generated when a local device received "change stream configuration" request.
```

Description

< This event is generated when a local device received "change stream configuration" request.

AVDTP_EVT_SEP_CAPABILITIES_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SEP_CAPABILITIES_RECEIVED 16    ///< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
```

Description

< This event is generated when a local device received a positive response to a "get SEP capabilities" request.

AVDTP_EVT_SEP_INFO_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SEP_INFO_RECEIVED 15    ///< This event is generated for each SEP contained in a positive response to a "discover" request.
```

Description

< This event is generated for each SEP contained in a positive response to a "discover" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED 6     ///< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED 50    ///< This event is generated when a local device received "set stream configuration" request.
```

Description

< This event is generated when a local device received "set stream configuration" request.

AVDTP_EVT_START_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_START_STREAM_COMPLETED 10    ///< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.

AVDTP_EVT_START_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_START_STREAM_REQUESTED 52    ///< This event is generated when a local device received "start stream" request.
```

Description

< This event is generated when a local device received "start stream" request.

AVDTP_EVT_STREAM_ABORTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_ABORTED 64    ///< This event is generated when a local device has successfully aborted a stream.
```

Description

< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.

AVDTP_EVT_STREAM_CLOSED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CLOSED 62    ///< This event is generated when a local device has successfully closed a stream.
```

Description

< This event is generated when a local device has successfully closed a stream. < This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.

AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED 17    ///< This event is generated when a local device received a positive response to a "get stream configuration" request.
```

Description

< This event is generated when a local device received a positive response to a "get stream configuration" request.

AVDTP_EVT_STREAM_CONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CONFIGURED 58    ///< This event is generated when a local device has successfully
```

configured a stream.

Description

< This event is generated when a local device has successfully configured a stream. < This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.

AVDTP_EVT_STREAM_OPENED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_OPENED 60    ///< This event is generated when a local device has successfully
opened a stream.
```

Description

< This event is generated when a local device has successfully opened a stream. < This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.

AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED 8     ///< This event is generated when a local device
received a response (either positive or negative) to a "change stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

AVDTP_EVT_STREAM_RECONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_RECONFIGURED 59    ///< This event is generated when a local device has
successfully reconfigured a stream.
```

Description

< This event is generated when a local device has successfully reconfigured a stream. < This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.

AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED 13    ///< This event is generated when a local device
received a response (either positive or negative) to a "exchange content protection control data" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.

AVDTP_EVT_STREAM_STARTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_STARTED 61    ///< This event is generated when a local device has successfully started a stream.
```

Description

< This event is generated when a local device has successfully started a stream. < This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.

AVDTP_EVT_STREAM_SUSPENDED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_SUSPENDED 63    ///< This event is generated when a local device has successfully suspended a stream.
```

Description

< This event is generated when a local device has successfully suspended a stream. < This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.

AVDTP_EVT_SUSPEND_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SUSPEND_STREAM_COMPLETED 12    ///< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.

AVDTP_EVT_SUSPEND_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SUSPEND_STREAM_REQUESTED 54    ///< This event is generated when a local device received "suspend stream" request.
```

Description

< This event is generated when a local device received "suspend stream" request.

AVDTP_MANAGER_STATE_CONNECTING Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_STATE_CONNECTING 1
```

Description

This is macro AVDTP_MANAGER_STATE_CONNECTING.

AVDTP_MANAGER_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVDTP_MANAGER_STATE_IDLE.

AVDTP_MAX_STREAM_TRANSPORT_SESSION Macro

File

avdtp.h

C

```
#define AVDTP_MAX_STREAM_TRANSPORT_SESSION 3
```

Description

This is macro AVDTP_MAX_STREAM_TRANSPORT_SESSION.

AVDTP_MEDIA_TYPE_AUDIO Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_AUDIO 0 ///< Audio.
```

Description

< Audio.

AVDTP_MEDIA_TYPE_MULTIMEDIA Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_MULTIMEDIA 2 ///< Both Audio & Video.
```

Description

< Both Audio & Video.

AVDTP_MEDIA_TYPE_VIDEO Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_VIDEO 1 ///< Video.
```

Description

< Video.

AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION 0x08 ///< Content Prortection. A SEP is capable of transferring content protection packets.
```

Description

< Content Prortection. A SEP is capable of transferring content protection packets.

AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION 0x10 ///< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
```

Description

< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC 0x40 ///< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
```

Description

< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT 0x01 ///< Media. A SEP is capable of transferring media (audio, video or both) packets.
```

Description

< Media. A SEP is capable of transferring media (audio, video or both) packets.

AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING 0x20 ///< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
```

Description

< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

AVDTP_SEP_CAPABILITY_FLAG_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_RECOVERY 0x04 ///< Recovery. A SEP is capable of transferring recovery packets.
```

Description

< Recovery. A SEP is capable of transferring recovery packets.

AVDTP_SEP_CAPABILITY_FLAG_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_REPORTING 0x02 ///< Reporting. A SEP is capable of transferring reporting packets.
```

Description

< Reporting. A SEP is capable of transferring reporting packets.

AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION 4 ///< Content Prortection. A SEP is capable of transferring content protection packets.
```

Description

< Content Prortection. A SEP is capable of transferring content protection packets.

AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION 5 ///< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
```

Description

< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC 7 ///< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
```

Description

< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT 1 ///< Media. A SEP is capable of transferring media (audio, video or both) packets.
```

Description

< Media. A SEP is capable of transferring media (audio, video or both) packets.

AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING 6 ///< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
```

Description

< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY 3 ///< Recovery. A SEP is capable of transferring recovery packets.
```

Description

< Recovery. A SEP is capable of transferring recovery packets.

AVDTP_SEP_SERVICE_CAPABILITY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_REPORTING 2 ///< Reporting. A SEP is capable of transferring reporting packets.
```

Description

< Reporting. A SEP is capable of transferring reporting packets.

AVDTP_SEP_STATE_FREE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_STATE_FREE 0
```

Description

This is macro AVDTP_SEP_STATE_FREE.

AVDTP_SEP_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_STATE_IDLE 1
```

Description

This is macro AVDTP_SEP_STATE_IDLE.

AVDTP_SEP_TYPE_SINK Macro

File

avdtp.h

C

```
#define AVDTP_SEP_TYPE_SINK 1 ///< Sink (usually a device like a headphones or BMW).
```

Description

< Sink (usually a device like a headphones or BMW).

AVDTP_SEP_TYPE_SOURCE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_TYPE_SOURCE 0 ///< Source (usually a device like a phone, desktop or laptop).
```

Description

< Source (usually a device like a phone, desktop or laptop).

AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS 2      ///< The stream is closing transport channels.
```

Description

< The stream is closing transport channels.

AVDTP_STREAM_FLAG_LISTENING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_FLAG_LISTENING 1
```

Description

This is macro AVDTP_STREAM_FLAG_LISTENING.

AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS 1      ///< The stream is opening transport channels.
```

Description

< The stream is opening transport channels.

AVDTP_STREAM_STATE_ABORTING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_ABORTING 7      ///< The stream is aborting. This means that all transport
channels associated with the stream are being closed.
```

Description

< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

AVDTP_STREAM_STATE_CLOSING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_CLOSING 6      ///< The stream is closing. This means that all transport
channels associated with the stream are being closed.
```

Description

< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

AVDTP_STREAM_STATE_CONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_CONFIGURED 3      ///< The stream has been configured.
```

Description

< The stream has been configured.

AVDTP_STREAM_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_IDLE 0      ///< The stream is idle. This can mean two things. The stream
specified by strm_handle does not exist
```

Description

< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.

AVDTP_STREAM_STATE_OPEN Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_OPEN 4      ///< The stream has been opened.
```

Description

< The stream has been opened.

AVDTP_STREAM_STATE_STREAMING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_STREAMING 5      ///< The stream has been started. Depending on the local SEP
type (source or sink) it means
```

Description

< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.

AVDTP_TC_OPCODE_CONNECT Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_OPCODE_CONNECT 0
```

Description

This is macro AVDTP_TC_OPCODE_CONNECT.

AVDTP_TC_OPCODE_DISCONNECT Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_OPCODE_DISCONNECT 1
```

Description

This is macro AVDTP_TC_OPCODE_DISCONNECT.

AVDTP_TC_STATUS_ERROR Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_STATUS_ERROR 0xFF
```

Description

This is macro AVDTP_TC_STATUS_ERROR.

AVDTP_TC_STATUS_SUCCESS Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_STATUS_SUCCESS 0x00
```

Description

This is macro AVDTP_TC_STATUS_SUCCESS.

AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED 0
```

Description

This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.

AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED 1
```

Description

This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.

AVDTP_TRANSPORT_SESSION_TYPE_MEDIA Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_MEDIA 0 ///< Media (audio or video).
```

Description

< Media (audio or video).

AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY 2 ///< Recovery (currently not supported).
```

Description

< Recovery (currently not supported).

AVDTP_TRANSPORT_SESSION_TYPE_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_REPORTING 1 ///< Reporting (currently not supported).
```

Description

< Reporting (currently not supported).

bt_avdtp_connect Macro

File

avdtp.h

C

```
#define bt_avdtp_connect(mgr, remote_addr) _bt_avdtp_open_control_channel_ex(mgr, remote_addr,
HCI_CONFIG_ENABLE_AUTHENTICATION | HCI_CONFIG_ENABLE_ENCRYPTION)
```

Description

brief Connect to a remote device. ingroup avdtp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr AVDTP manager. param remote_addr The address of a remote device.

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_avdtp_connect_ex Macro

File

avdtp.h

C

```
#define bt_avdtp_connect_ex(mgr, remote_addr, acl_config) _bt_avdtp_open_control_channel_ex(mgr,
remote_addr, acl_config)
```

Description

brief Connect to a remote device. ingroup avdtp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr AVDTP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values: li HCI_CONFIG_ENABLE_AUTHENTICATION li HCI_CONFIG_ENABLE_ENCRYPTION li HCI_CONFIG_BECOME_MASTER

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

AVDTP_CMD_DELAYREPORT Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_DELAYREPORT 13
```

Description

This is macro AVDTP_CMD_DELAYREPORT.

AVDTP_CMD_GET_ALL_CAPABILITIES Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_ALL_CAPABILITIES 12
```

Description

This is macro AVDTP_CMD_GET_ALL_CAPABILITIES.

AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T Macro

File

avdtp.h

C

```
#define AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T 2
```

Description

This is macro AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T.

AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT 1
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT.

AVDTP_SCMS_T_CP_BIT Macro

File

avdtp.h

C

```
#define AVDTP_SCMS_T_CP_BIT 2
```

Description

This is macro AVDTP_SCMS_T_CP_BIT.

AVDTP_SCMS_T_L_BIT Macro

File

avdtp.h

C

```
#define AVDTP_SCMS_T_L_BIT 1
```

Description

This is macro AVDTP_SCMS_T_L_BIT.

bt_avdtp_evt_delay_report_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_delay_report_completed_t {
  bt_byte err_code;
} bt_avdtp_evt_delay_report_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.

Description

brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.

bt_cp_header_t Type

File

avdtp.h

C

```
typedef struct _bt_cp_header_s bt_cp_header_t;
```

Description

This is type bt_cp_header_t.

_bt_cp_header_s Structure

File

avdtp.h

C

```
struct _bt_cp_header_s {
  union {
    bt_byte scmst;
    struct {
      bt_byte* value;
      bt_uint length;
    } unknown;
  } header;
};
```

Description

This is record _bt_cp_header_s.

bt_avdtp_evt_ctrl_connection_failed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_connection_failed_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_connection_failed_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.

bt_avdtp_evt_set_stream_configuration_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_byte int_sep_id;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
  bt_byte src_evt_code;
} bt_avdtp_evt_set_stream_configuration_t;
```

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.

AVDTP_EVT_DELAYREPORT_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_DELAYREPORT_COMPLETED 18    ///< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION 19
```

Description

This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.

AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING 0x80 ///< Delat reporitng.
```

Description

< Delat reporitng.

AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING 8 ///< Delay Reporting.
```

Description

< Delay Reporting.

AVRCP Functions

_bt_avrcp_allocate_browsing_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_browsing_cmd(bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_browsing_cmd.

_bt_avrcp_allocate_bt_specific_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_bt_specific_cmd(bt_byte ctype, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_bt_specific_cmd.

_bt_avrcp_allocate_bt_specific_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_bt_specific_response(bt_byte ctype, bt_byte tran_id, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_bt_specific_response.

_bt_avrcp_allocate_channel Function

File

avrcp_private.h

C

```
bt_avrcp_channel_t* _bt_avrcp_allocate_channel(bt_avrcp_mgr_t* mgr);
```

Description

void _bt_avctp_l2cap_read_data_callback(struct _channel *pch, bt_byte* pdata, bt_int len); bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint profile_id);

_bt_avrcp_allocate_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_cmd();
```

Description

This is function _bt_avrcp_allocate_cmd.

_bt_avrcp_allocate_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_response(bt_byte tran_id);
```

Description

This is function _bt_avrcp_allocate_response.

_bt_avrcp_allocate_simple_panel_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_simple_panel_cmd(bt_byte ctype, bt_byte opid, bt_byte button_state);
```

Description

This is function _bt_avrcp_allocate_simple_panel_cmd.

_bt_avrcp_allocate_simple_panel_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_simple_panel_response(bt_byte ctype, bt_byte tran_id, bt_byte opid,
bt_byte button_state);
```

Description

This is function _bt_avrcp_allocate_simple_panel_response.

_bt_avrcp_find_channel Function

File

avrcp_private.h

C

```
bt_avrcp_channel_t* _bt_avrcp_find_channel(bt_avrcp_mgr_t* mgr, bt_avctp_channel_t* avctp_channel);
```

Description

This is function _bt_avrcp_find_channel.

_bt_avrcp_free_channel Function

File

avrcp_private.h

C

```
void _bt_avrcp_free_channel(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_free_channel.

_bt_avrcp_free_cmd Function

File

avrcp_private.h

C

```
void _bt_avrcp_free_cmd(bt_av_command_t* command);
```

Description

This is function _bt_avrcp_free_cmd.

_bt_avrcp_get_tick_count Function

File

avrcp_private.h

C

```
bt_long _bt_avrcp_get_tick_count();
```

Description

This is function _bt_avrcp_get_tick_count.

_bt_avrcp_handle_add_to_now_playing Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_add_to_now_playing(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint
len);
```

Description

This is function _bt_avrcp_handle_add_to_now_playing.

_bt_avrcp_handle_command Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_command(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_received_t* evt_param);
```

Description

This is function _bt_avrcp_handle_command.

_bt_avrcp_handle_command_sent Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_command_sent(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_sent_t* evt_param);
```

Description

This is function _bt_avrcp_handle_command_sent.

_bt_avrcp_handle_get_capabilities Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_capabilities(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint
len);
```

Description

This is function _bt_avrcp_handle_get_capabilities.

_bt_avrcp_handle_get_current_player_application_setting_value Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_current_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte
ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_current_player_application_setting_value.

_bt_avrcp_handle_get_element_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_element_attributes(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_get_element_attributes.

_bt_avrcp_handle_get_play_status Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_play_status(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint
len);
```

Description

This is function _bt_avrcp_handle_get_play_status.

_bt_avrcp_handle_get_player_application_setting_attribute_text Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_player_application_setting_attribute_text(bt_avrcp_channel_t* channel, bt_byte
ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_player_application_setting_attribute_text.

_bt_avrcp_handle_get_player_application_setting_value_text Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_player_application_setting_value_text(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_player_application_setting_value_text.

_bt_avrcp_handle_inform_battery_status_of_ct Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_inform_battery_status_of_ct(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte*
data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_inform_battery_status_of_ct.

_bt_avrcp_handle_inform_displayable_character_set Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte*
data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_inform_displayable_character_set.

_bt_avrcp_handle_list_player_application_setting_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_list_player_application_setting_attributes(bt_avrcp_channel_t* channel, bt_byte
ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_list_player_application_setting_attributes.

_bt_avrcp_handle_list_player_application_setting_values Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_list_player_application_setting_values(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_list_player_application_setting_values.

_bt_avrcp_handle_play_item Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_play_item(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_play_item.

_bt_avrcp_handle_register_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_register_notification(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte event_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_register_notification.

_bt_avrcp_handle_request_continuing_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_request_continuing_response(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte*
data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_request_continuing_response.

_bt_avrcp_handle_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_response(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_received_t* evt_param);
```

Description

This is function _bt_avrcp_handle_response.

_bt_avrcp_handle_response_sent Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_response_sent(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_sent_t* evt_param);
```

Description

This is function _bt_avrcp_handle_response_sent.

_bt_avrcp_handle_set_absolute_volume Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_set_absolute_volume.

_bt_avrcp_handle_set_addressed_player Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_addressed_player(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_set_addressed_player.

_bt_avrcp_handle_set_player_application_setting_value Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_set_player_application_setting_value.

_bt_avrcp_init_signal Function

File

avrcp_private.h

C

```
void _bt_avrcp_init_signal();
```

Description

void _bt_avctp_init_message_buffers();

_bt_avrcp_init_timer Function

File

avrcp_private.h

C

```
void _bt_avrcp_init_timer();
```

Description

This is function _bt_avrcp_init_timer.

_bt_avrcp_register_next_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_register_next_notification(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_register_next_notification.

_bt_avrcp_register_pending_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_register_pending_notification(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_register_pending_notification.

_bt_avrcp_send_notifications Function

File

avrcp_private.h

C

```
void _bt_avrcp_send_notifications(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_send_notifications.

_bt_avrcp_set_signal Function

File

avrcp_private.h

C

```
void _bt_avrcp_set_signal();
```

Description

This is function _bt_avrcp_set_signal.

_bt_avrcp_start_timer Function

File

avrcp_private.h

C

```
void _bt_avrcp_start_timer();
```

Description

This is function _bt_avrcp_start_timer.

_bt_avrcp_tg_handle_get_capabilities Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_capabilities(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_capabilities.

_bt_avrcp_tg_handle_get_element_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_element_attributes(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte
tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_element_attributes.

_bt_avrcp_tg_handle_get_play_status Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_play_status(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_play_status.

_bt_avrcp_tg_handle_inform_battery_status_of_ct Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_inform_battery_status_of_ct(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte
tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_inform_battery_status_of_ct.

_bt_avrcp_tg_handle_inform_displayable_character_set Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_inform_displayable_character_set.

_bt_avrcp_tg_handle_register_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_register_notification(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_register_notification.

_bt_avrcp_tg_handle_set_absolute_volume Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_set_absolute_volume.

_bt_avrcp_write_command_header Function

File

avrcp_private.h

C

```
bt_int _bt_avrcp_write_command_header(bt_av_command_t* command);
```

Description

This is function _bt_avrcp_write_command_header.

bt_avrcp_abort_continuing_response Function

File

avrcp_private.h

C

```
bt_bool bt_avrcp_abort_continuing_response(bt_avrcp_channel_t* channel, bt_byte pduid);
```

Description

This is function bt_avrcp_abort_continuing_response.

bt_avrcp_add_to_now_playing Function

File

avrcp.h

C

```
bt_bool bt_avrcp_add_to_now_playing(bt_avrcp_channel_t* channel, bt_byte scope, bt_av_element_id_t*
element_id, bt_uint counter);
```

Description

brief Add to "now playing" list. ingroup avrcp

details This function adds a media element specified by p element_id to the "now playing" list on the target.

param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING

param element_id UID of the media element to be added to the "now playing" list. param counter UID counter.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_avrcpt_request_continuing_response Function

File

avrcp_private.h

C

```
bt_bool bt_avrcp_avrcpt_request_continuing_response(bt_avrcp_channel_t* channel, bt_byte pduid);
```

Description

This is function bt_avrcp_avrcpt_request_continuing_response.

bt_avrcp_cancel_find Function

File

avrcp.h

C

```
bt_bool bt_avrcp_cancel_find();
```

Description

brief Cancel finding Targets ingroup avrcp

details This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_EVT_SEARCH_COMPLETED event will be generated.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_cancel_listen Function

File

avrcp.h

C

```
void bt_avrcp_cancel_listen(bt_avrcp_channel_t* channel);
```

Description

brief Cancel listening for incoming connections. ingroup avrcp

details This function stops listening for incoming connections on a specified channel.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_connect Function

File

avrcp.h

C

```
bt_bool bt_avrcp_connect(bt_avrcp_channel_t* channel, bt_bdaddr_t* remote_address);
```

Description

brief Connect to a remote device. ingroup avrcp

details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_EVT_CONTROL_CONNECTION_FAILED.

param channel AVRCP channel. param remote_address The address of a remote device.

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_avrcp_create_channel Function

File

avrcp.h

C

```
bt_avrcp_channel_t* bt_avrcp_create_channel(bt_avrcp_mgr_t* mgr, bt_bool create_browsing_channel);
```

Description

brief Allocate AVRCP channel ingroup avrcp

details This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel.

param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created.

return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.

bt_avrcp_create_outgoing_channel Function

File

avrcp.h

C

```
bt_avrcp_channel_t* bt_avrcp_create_outgoing_channel(bt_avrcp_mgr_t* mgr, bt_bool create_browsing_channel);
```

Description

brief Allocate AVRCP channel ingroup avrcp

details This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels.

param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created.

return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.

bt_avrcp_destroy_channel Function

File

avrcp.h

C

```
bt_bool bt_avrcp_destroy_channel(bt_avrcp_channel_t* channel);
```

Description

brief Destroy AVRCP channel. ingroup avrcp

details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_disconnect Function

File

avrcp.h

C

```
bt_bool bt_avrcp_disconnect(bt_avrcp_channel_t* channel);
```

Description

brief Disconnect from a remote device. ingroup avrcp

details This function closes a connection to a remote device.

param channel AVRCP channel.

return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

bt_avrcp_find_controller Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_controller(bt_bdaddr_t* deviceAddress, bt_avrcp_find_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find Controller ingroup avrcp

details This function looks for a controller on a remote device specified by p deviceAddress.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p callback and p client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_find_target Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_target(bt_bdaddr_t* deviceAddress, bt_avrcp_find_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find Target ingroup avrcp

details This function looks for a target on a remote device specified by p deviceAddress.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p callback and p client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_find_targets Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_targets(bt_byte search_length);
```

Description

brief Find Targets ingroup avrcp

details This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed.

param search_length The amount of time the search will be performed for.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_get_browsing_channel_state Function

File

avrcp.h

C

```
bt_byte bt_avrcp_get_browsing_channel_state(bt_avrcp_channel_t* channel);
```

Description

brief Get AVCTP browsing channel state. ingroup avrcp

details This function returns status of the AVCTP browsing channel.

param channel AVRCP channel.

return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avrcp_get_channel_remote_address Function

File

avrcp.h

C

```
bt_bdaddr_t* bt_avrcp_get_channel_remote_address(bt_avrcp_channel_t* channel);
```

Description

brief Get channel's remote BT address. ingroup avrcp

details This function returns the address of the remote device associated with the channel.

param channel AVRCP channel.

return li The address of the remote device if channel is connected. li NULL otherwise.

bt_avrcp_get_company_id_list Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_company_id_list(bt_avrcp_channel_t* channel);
```

Description

brief Get Company ID list. ingroup avrcp

details This function requests a list of company id's supported by the remote device

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_control_channel_state Function

File

avrcp.h

C

```
bt_byte bt_avrcp_get_control_channel_state(bt_avrcp_channel_t* channel);
```

Description

brief Get AVCTP control channel state. ingroup avrcp

details This function returns status of the AVCTP control channel.

param channel AVRCP channel.

return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avrcp_get_current_player_application_setting_value Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_current_player_application_setting_value(bt_avrcp_channel_t* channel,
bt_av_player_setting_current_values_t* response_buffer);
```

Description

brief Get current player setting values. ingroup avrcp

details This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_current_values_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_element_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_element_attributes(bt_avrcp_channel_t* channel, bt_av_element_id_t* element_id,
bt_uint attr_mask);
```

Description

brief Get media element attributes. ingroup avrcp

details This function requests the attributes of the element specified with p element_id.

note Currently p element_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track.

param channel AVRCP channel. param element_id UID of the media element whose attributes are requested. param attr_mask Bitmask that defines which attributes are requested. This value can be a combination of the following values: li AVC_MEDIA_ATTR_FLAG_TITLE li AVC_MEDIA_ATTR_FLAG_ARTIST li AVC_MEDIA_ATTR_FLAG_ALBUM li AVC_MEDIA_ATTR_FLAG_NUMBER li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER li AVC_MEDIA_ATTR_FLAG_GENRE li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME li AVC_MEDIA_ATTR_FLAG_ALL

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_hci_connection Function

File

avrcp.h

C

```
bt_hci_conn_state_p* bt_avrcp_get_hci_connection(bt_avrcp_channel_t* channel);
```

Description

brief Get HCI connection for a channel ingroup avrcp

details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect.

param channel AVRCP channel.

return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does not exist or there is no HCI connection between local and remote devices associated with the channel.

bt_avrcp_get_mgr Function

File

avrcp.h

C

```
bt_avrcp_mgr_t* bt_avrcp_get_mgr();
```

Description

brief Return a pointer to an instance of the AVRCP manager. ingroup avrcp

details This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.

bt_avrcp_get_play_status Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_play_status(bt_avrcp_channel_t* channel, bt_uint repeat_interval);
```

Description

brief Get playback status. ingroup avrcp

details This function requests the status of the currently playing media at the target.

param channel AVRCP channel. param repeat_interval Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_player_application_setting_attr_text Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_player_application_setting_attr_text(bt_avrcp_channel_t* channel,
bt_av_player_settings_text_t* response_buffer);
```

Description

brief Get player setting attribute text. ingroup avrcp

details This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_settings_text_t::setting_id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_text_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_player_application_setting_value_text Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_player_application_setting_value_text(bt_avrcp_channel_t* channel, bt_byte attr_id,
bt_av_player_setting_values_text_t* response_buffer);
```

Description

brief Get player setting value text. ingroup avrcp

details This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_text_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_subuint_info Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_get_subuint_info(bt_avrcp_channel_t* channel);
```

Description

brief Get subunit info ingroup avrcp

details This function is used to request subunit info from the target.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_supported_event_id_list Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_supported_event_id_list(bt_avrcp_channel_t* channel);
```

Description

brief Get supported events. ingroup avrcp

details This function requests a list of events supported by the remote device

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_unit_info Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_get_unit_info(bt_avrcp_channel_t* channel);
```

Description

brief Get unit info ingroup avrcp

details This function is used to request unit info from the target.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_inform_battery_status Function

File

avrcp.h

C

```
bt_bool bt_avrcp_inform_battery_status(bt_avrcp_channel_t* channel, bt_byte status);
```

Description

brief Inform controller's battery status. ingroup avrcp

details This function is used to inform the target about the controller's battery status.

param channel AVRCP channel. param status Battery status. This can be one of the following values: li AVC_BATTERY_STATUS_NORMAL li AVC_BATTERY_STATUS_WARNING li AVC_BATTERY_STATUS_CRITICAL li AVC_BATTERY_STATUS_EXTERNAL li AVC_BATTERY_STATUS_FULL_CHARGE

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_inform_displayable_character_set Function

File

avrcp.h

C

```
bt_bool bt_avrcp_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_uint* charset_list,
bt_byte charset_count);
```

Description

brief Inform displayable character set. ingroup avrcp

details This function informs the list of character set supported by the controller to the target.

param channel AVRCP channel. param charset_list List of displayable character sets. param charset_count Number of entries in the list.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_init_controller Function

File

avrcp.h

C

```
void bt_avrcp_init_controller();
```

Description

brief Initialize AVRCP to be used in controller mode. ingroup avrcp

details This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

bt_avrcp_init_target Function

File

avrcp.h

C

```
void bt_avrcp_init_target(bt_ulong company_id, bt_uint supported_events);
```

Description

brief Initialize AVRCP to be used in target mode. ingroup avrcp

details This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called.

param company_id The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used. param supported_events Bitmask that specifies events supported by the target. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED

bt_avrcp_list_player_application_setting_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_list_player_application_setting_attributes(bt_avrcp_channel_t* channel,
bt_av_player_settings_t* response_buffer);
```

Description

brief Get supported player setting attributes. ingroup avrcp

details This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings_t::count to the number of entries in the list.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_list_player_application_setting_values Function

File

avrcp.h

C

```
bt_bool bt_avrcp_list_player_application_setting_values(bt_avrcp_channel_t* channel, bt_byte attr_id,
bt_av_player_setting_values_t* response_buffer);
```

Description

brief Get player setting attribute values. ingroup avrcp

details This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_listen Function

File

avrcp.h

C

```
bt_bool bt_avrcp_listen(bt_avrcp_channel_t* channel);
```

Description

brief Listen for incoming connections. ingroup avrcp

details This function enables incoming connections on the specified AVRCP channel.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_play_item Function

File

avrcp.h

C

```
bt_bool bt_avrcp_play_item(bt_avrcp_channel_t* channel, bt_byte scope, bt_av_element_id_t* element_id,
bt_uint counter);
```

Description

brief Play media item. ingroup avrcp

details This function starts playing an item indicated by the UID.

param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be played. param counter UID counter.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_register_notification Function

File

avrcp.h

C

```
bt_bool bt_avrcp_register_notification(bt_avrcp_channel_t* channel, bt_byte event_id, bt_ulong
playback_interval);
```

Description

brief Register notification. ingroup avrcp

details This function registers with the target to receive notifications asynchronously based on specific events occurring.

param channel AVRCP channel. param event_id Event Id. This value can be one of the following values: li AVC_EVENT_PLAYBACK_STATUS_CHANGED li AVC_EVENT_TRACK_CHANGED li AVC_EVENT_TRACK_REACHED_END li AVC_EVENT_TRACK_REACHED_START li AVC_EVENT_PLAYBACK_POS_CHANGED li AVC_EVENT_BATT_STATUS_CHANGED li AVC_EVENT_SYSTEM_STATUS_CHANGED li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_UIDS_CHANGED li AVC_EVENT_VOLUME_CHANGED param playback_interval The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_register_notifications Function

File

avrcp.h

C

```
bt_bool bt_avrcp_register_notifications(bt_avrcp_channel_t* channel, bt_uint event_mask);
```

Description

brief Register notifications. ingroup avrcp

details This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call.

note This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event.

param channel AVRCP channel. param event_mask Bitmask that specifies which events to register for. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED li AVC_EVENT_FLAG_ALL

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_button_click Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_button_click(bt_avrcp_channel_t* channel, bt_byte button_id);
```

Description

brief Send AV/C Panel Subunit "click" PASS THROUGH command ingroup avrcp

details This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PANEL_BUTTON_RELEASED

param channel AVRCP channel. param button_id Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_cmd Function

File

avrcp.h

C

```
bt_bool bt_avrcp_send_cmd(bt_avrcp_channel_t* channel, bt_av_command_t* command);
```

Description

brief Send AVRCP command ingroup avrcp

details This function sends a command to the remote device.

param channel AVRCP channel. param command Command to be sent.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_panel_control(bt_avrcp_channel_t* channel, bt_byte opid, bt_byte button_state);
```

Description

brief Send AV/C Panel Subunit "control" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_CTYPE_CONTROL.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_press_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_press_panel_control(bt_avrcp_channel_t* channel, bt_byte opid);
```

Description

brief Send AV/C Panel Subunit "pressed" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_PRESSED.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_simple_panel_cmd Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_simple_panel_cmd(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte opid, bt_byte button_state);
```

Description

brief Send AV/C Panel Subunit PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command to the target.

param channel AVRCP channel. param ctype Command type. This value can be on of the following values: li AVC_CTYPE_CONTROL 0 li AVC_CTYPE_STATUS 1 li AVC_CTYPE_SPECIFIC_IQUIRY 2 li AVC_CTYPE_NOTIFY 3 li AVC_CTYPE_GENERAL_INQUORY 4 param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_absolute_volume Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte volume);
```

Description

brief Set absolute volume. ingroup avrcp

details This function is used to set an absolute volume to be used by the rendering device.

param channel AVRCP channel. param volume Volume

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_addressed_player Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_addressed_player(bt_avrcp_channel_t* channel, bt_uint player_id);
```

Description

brief Set addressed player. ingroup avrcp

details This function is used to inform the target of which media player the controller wishes to control.

param channel AVRCP channel. param player_id Player Id.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_player_application_setting_value Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte* attr_id_list, bt_byte* attr_value_list, bt_byte attr_id_count);
```

Description

brief Set player setting attribute values. ingroup avrcp

details This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

param channel AVRCP channel. param attr_id_list List of setting attribute ids. param attr_value_list List of setting attribute value ids. param attr_id_count The number of entries in both lists.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_start Function

File

avrcp.h

C

```
void bt_avrcp_start(bt_avrcp_mgr_t* mgr, bt_avrcp_mgr_callback_fp callback, void* callback_param);
```

Description

brief Start the AVRCP layer. ingroup avrcp

details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the c callback and c callback_param in the bt_avrcp_mgr_t structure.

param mgr AVRCP manager. param callback The callback function that will be called when the AVRCP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_avrcp_tg_send_element_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_tg_send_element_attributes(bt_avrcp_channel_t* channel, bt_byte tran_id,
bt_av_element_attribute_t* attrs, bt_byte attr_count);
```

Description

brief Send media element attributes ingroup avrcp

details This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

param status Battery status

bt_avrcp_tg_set_absolute_volume Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_absolute_volume(bt_byte volume);
```

Description

brief Set absolute volume ingroup avrcp

details This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified.

param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.

bt_avrcp_tg_set_battery_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_battery_status(bt_byte status);
```

Description

brief Set battery status ingroup avrcp

details This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified.

param status Battery status

bt_avrcp_tg_set_channel_absolute_volume Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_channel_absolute_volume(bt_avrcp_channel_t* channel, bt_byte volume);
```

Description

This is function bt_avrcp_tg_set_channel_absolute_volume.

bt_avrcp_tg_set_current_track Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_current_track(bt_av_element_id_t* track_id, bt_ulong song_length, bt_ulong
song_position);
```

Description

brief Set current track ingroup avrcp

details This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified.

param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.

bt_avrcp_tg_set_play_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_play_status(bt_ulong song_length, bt_ulong song_position, bt_byte play_status);
```

Description

local target control *

- brief Set playback status
- ingroup avrcp

*

- details This function is used to set the playback status when AVRCP is running in target mode.
- If there are active connections with controllers, the ones that registered
- for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified.

*

- param song_length The length of the current track.
- param song_position The position of the current track.
- param play_status Playback status. This value can be one of the following values:
- li AVC_PLAY_STATUS_STOPPED
- li AVC_PLAY_STATUS_PLAYING
- li AVC_PLAY_STATUS_PAUSED

- li AVC_PLAY_STATUS_FW_SEEK
- li AVC_PLAY_STATUS_REV_SEEK

bt_avrcp_tg_set_system_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_system_status(bt_byte status);
```

Description

brief Set system status ingroup avrcp

details This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

param status Battery status

_bt_avrcp_allocate_browsing_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_browsing_response(bt_byte tran_id, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_browsing_response.

_bt_avrcp_general_reject Function

File

avrcp_private.h

C

```
bt_bool _bt_avrcp_general_reject(bt_avrcp_channel_t* channel, bt_byte reason, bt_byte tran_id);
```

Description

This is function _bt_avrcp_general_reject.

_bt_avrcp_send_rejected_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_send_rejected_response(bt_avrcp_channel_t* channel, bt_byte tran_id, bt_byte pdu_id);
```

Description

This is function _bt_avrcp_send_rejected_response.

bt_avrcp_send_release_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_release_panel_control(bt_avrcp_channel_t* channel, bt_byte opid);
```

Description

brief Send AV/C Panel Subunit "released" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_RELEASED.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

AVRCP Data Types and Constants

_bt_av_command_t Structure

File

avrcp_command.h

C

```
struct _bt_av_command_t {
  bt_av_command_t* next_command;
  bt_byte cmd_type;
  bt_byte flags;
  bt_byte ctype;
  bt_byte subunit_type;
  bt_byte subunit_id;
  bt_byte opcode;
  bt_byte pdu_id;
  bt_byte* parameters;
  bt_uint params_len;
  bt_av_response_t* response_buffer;
  bt_long send_time;
  bt_byte tran_id;
};
```

Description

This is record _bt_av_command_t.

_bt_avrcp_channel_t Structure

File

avrcp.h

C

```
struct _bt_avrcp_channel_t {
  bt_byte state;
  bt_byte flags;
  bt_uint cur_event_mask;
  bt_uint new_event_mask;
  bt_byte play_status;
  bt_uint req_play_status_interval;
  bt_ulong play_status_req_time;
  bt_avctp_channel_t* control_channel;
  bt_avctp_channel_t* browsing_channel;
  bt_avrcp_mgr_t* avrcp_mgr;
  bt_queue_element_t* send_cq_head;
  bt_queue_element_t* ack_cq_head;
  bt_uint tg_event_mask;
  bt_uint tg_interim_event_mask;
  bt_uint tg_signalled_event_mask;
  bt_ulong notify_song_position;
  bt_byte register_notification_tran_id[AVC_MAXEVENTS];
  bt_byte tg_next_event_id;
  bt_byte volume;
};
```

Members

Members	Description
bt_uint tg_event_mask;	target data

Description

brief AVRCP channel description ingroup avrcp

details This structure is used to hold information about an AVRCP channel.

_bt_avrcp_mgr_t Structure

File

avrcp.h

C

```
struct _bt_avrcp_mgr_t {
  bt_byte state;
  bt_byte flags;
  bt_ulong company_id;
  bt_uint supported_events;
  bt_ulong song_length;
  bt_ulong song_position;
  bt_byte play_status;
  bt_byte volume;
  bt_byte battery_status;
  bt_byte system_status;
  bt_av_element_id_t current_track_id;
  bt_uint uid_counter;
  bt_avrcp_channel_t* channels;
  bt_avrcp_mgr_callback_fp callback;
  void* callback_param;
};
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li AVRCP_MANAGER_STATE_IDLE < li AVRCP_MANAGER_STATE_CONNECTING < li AVRCP_MANAGER_STATE_DISCONNECTING
bt_byte flags;	< Additional manager state. This value can be a combination of the following values: < li AVRCP_MANAGER_FLAG_SEARCHING
bt_ulong company_id;	< The 24-bit unique ID obtained from the IEEE Registration Authority Committee. < If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used.
bt_uint supported_events;	< Events supported by the target. This value can be a combination of the following values: < li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED < li AVC_EVENT_FLAG_TRACK_CHANGED < li AVC_EVENT_FLAG_TRACK_REACHED_END < li AVC_EVENT_FLAG_TRACK_REACHED_START < li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED < li AVC_EVENT_FLAG_BATT_STATUS_CHANGED < li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED < li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED < li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED < li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED < li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED < li AVC_EVENT_FLAG_UIDS_CHANGED < li AVC_EVENT_FLAG_VOLUME_CHANGED
bt_ulong song_length;	< Current song length
bt_ulong song_position;	< Current song poistion
bt_byte play_status;	< Current playback status
bt_byte volume;	< Current volumne
bt_byte battery_status;	< Current battery statusThis can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE
bt_byte system_status;	< Current system status
bt_av_element_id_t current_track_id;	< Current track id

bt_uint uid_counter;	< The number of media items in the target.
bt_avrcp_channel_t* channels;	< List of available AVRCP channels.
bt_avrcp_mgr_callback_fp callback;	< Pointer to a function used to notify the AVRCP consumer about various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.

Description

brief AVRCP manager. ingroup avrcp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avrcp_get_mgr().

bt_av_add_to_now_playing_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_add_to_now_playing_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_add_to_now_playing_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of the request.

Description

brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.

bt_av_battery_status_of_ct_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_battery_status_of_ct_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_battery_status_of_ct_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< Battery status. This can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE

Description

brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.

bt_av_capability_company_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_capability_company_id_s {
  bt_av_response_t header;
  bt_byte count;
  bt_ulong* company_id_list;
} bt_av_capability_company_id_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported company ids.
bt_ulong* company_id_list;	< List of supported company ids.

Description

brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.

bt_av_capability_event_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_capability_event_id_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* event_id_list;
} bt_av_capability_event_id_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported events ids.
bt_byte* event_id_list;	< List of supported event ids.

Description

brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.

bt_av_command_t Type

File

avrcp_command.h

C

```
typedef struct _bt_av_command_t bt_av_command_t;
```

Description

This is type bt_av_command_t.

bt_av_displayable_character_set_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_displayable_character_set_s {
```

```
  bt_av_response_t header;
  bt_byte count;
  bt_uint* charset_list;
} bt_av_displayable_character_set_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported characters sets by the controller.
bt_uint* charset_list;	< List of supported characters sets by the controller.

Description

brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.

bt_av_element_attribute_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_attribute_s {
  bt_ulong id;
  bt_uint charset;
  bt_uint len;
  bt_byte* value;
} bt_av_element_attribute_t;
```

Members

Members	Description
bt_ulong id;	< Attribute Id.
bt_uint charset;	< Charcater set.
bt_uint len;	< Value length.
bt_byte* value;	< Attribute value.

Description

brief Media element attribute ingroup avrcp

details This structure is used to store media element attribute.

bt_av_element_attributes_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_attributes_s {
  bt_av_response_t header;
  bt_byte count;
  bt_av_element_attribute_t* attr_list;
} bt_av_element_attributes_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of attributes returned.
bt_av_element_attribute_t* attr_list;	< List of attribute values

Description

brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union -

bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.

bt_av_element_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_id_s {
  bt_ulong id_lo;
  bt_ulong id_hi;
} bt_av_element_id_t;
```

Members

Members	Description
bt_ulong id_lo;	< 4 least significant bytes of UID.
bt_ulong id_hi;	< 4 most significant bytes of UID.

Description

brief Media element UID ingroup avrcp

details This structure is used to store media element UID.

bt_av_get_element_attributes_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_get_element_attributes_s {
  bt_av_response_t header;
  bt_av_element_id_t id;
  bt_byte attributes;
} bt_av_get_element_attributes_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_av_element_id_t id;	< Media element UID
bt_byte attributes;	< Bitmask that defines attributes requested. This can be a combination of the following values: < li AVC_MEDIA_ATTR_FLAG_TITLE < li AVC_MEDIA_ATTR_FLAG_ARTIST < li AVC_MEDIA_ATTR_FLAG_ALBUM < li AVC_MEDIA_ATTR_FLAG_NUMBER < li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER < li AVC_MEDIA_ATTR_FLAG_GENRE < li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME

Description

brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.

bt_av_notification_addressed_player_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_addressed_player_changed_s {
  bt_uint player_id;
  bt_uint uid_counter;
} bt_av_notification_addressed_player_changed_t;
```

Members

Members	Description
bt_uint player_id;	< New player Id
bt_uint uid_counter;	< UID counter

Description

brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.

bt_av_notification_app_setting_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_app_setting_changed_s {
  bt_byte setting_id;
  bt_byte setting_value_id;
} bt_av_notification_app_setting_changed_t;
```

Description

This is type bt_av_notification_app_setting_changed_t.

bt_av_notification_battery_status_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_battery_status_s {
  bt_byte status;
} bt_av_notification_battery_status_t;
```

Members

Members	Description
bt_byte status;	< New battery status.This can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE

Description

brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.

bt_av_notification_playback_pos_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_playback_pos_changed_s {
  bt_ulong position;
} bt_av_notification_playback_pos_changed_t;
```

Members

Members	Description
bt_ulong position;	< New playback position

Description

brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.

bt_av_notification_playback_status_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_playback_status_changed_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_notification_playback_status_changed_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< Play status. This can be on of the following values: < li AVC_PLAY_STATUS_STOPPED < li AVC_PLAY_STATUS_PLAYING < li AVC_PLAY_STATUS_PAUSED < li AVC_PLAY_STATUS_FW_SEEK < li AVC_PLAY_STATUS_REV_SEEK < li AVC_PLAY_STATUS_ERROR

Description

brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.

bt_av_notification_system_status_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_system_status_changed_s {
  bt_byte status;
} bt_av_notification_system_status_changed_t;
```

Members

Members	Description
bt_byte status;	< New system status

Description

brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.

bt_av_notification_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_s {
  bt_av_response_t header;
  union {
    bt_av_notification_playback_status_changed_t play_status;
    bt_av_notification_track_changed_t track;
    bt_av_notification_playback_pos_changed_t playback_pos;
```

```
    bt_av_notification_battery_status_t battery_status;
    bt_av_notification_system_status_changed_t system_status;
    bt_av_notification_addressed_player_changed_t addressed_player;
    bt_av_notification_uids_changed_t uids;
    bt_av_notification_volume_changed_t volume;
    bt_av_notification_app_setting_changed_t app_setting;
  } params;
} bt_av_notification_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_av_notification_playback_status_changed_t play_status;	< Valid if notification is AVRCP_EVT_PLAYBACK_STATUS_CHANGED
bt_av_notification_track_changed_t track;	< Valid if notification is AVRCP_EVT_TRACK_CHANGED
bt_av_notification_playback_pos_changed_t playback_pos;	< Valid if notification is AVRCP_EVT_PLAYBACK_POS_CHANGED
bt_av_notification_battery_status_t battery_status;	< Valid if notification is AVRCP_EVT_BATT_STATUS_CHANGED
bt_av_notification_system_status_changed_t system_status;	< Valid if notification is AVRCP_EVT_SYSTEM_STATUS_CHANGED
bt_av_notification_addressed_player_changed_t addressed_player;	< Valid if notification is AVRCP_EVT_ADDRESSED_PLAYER_CHANGED
bt_av_notification_uids_changed_t uids;	< Valid if notification is AVRCP_EVT_UIDS_CHANGED
bt_av_notification_volume_changed_t volume;	< Valid if notification is AVRCP_EVT_VOLUME_CHANGED
bt_av_notification_app_setting_changed_t app_setting;	< Valid if notification is AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED

Description

brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed The notification code defines which member of the bt_av_notification_t::params union is valid @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED c bt_av_notification_playback_status_changed_t play_status @arg AVRCP_EVT_TRACK_CHANGED c bt_av_notification_track_changed_t track @arg AVRCP_EVT_PLAYBACK_POS_CHANGED c bt_av_notification_playback_pos_changed_t playback_pos @arg AVRCP_EVT_BATT_STATUS_CHANGED c bt_av_notification_battery_status_t battery_status @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED c bt_av_notification_system_status_changed_t system_status @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED c bt_av_notification_addressed_player_changed_t addressed_player @arg AVRCP_EVT_UIDS_CHANGED c bt_av_notification_uids_changed_t uids @arg AVRCP_EVT_VOLUME_CHANGED c bt_av_notification_volume_changed_t volume @arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED c bt_av_notification_app_setting_changed_t app_setting

bt_av_notification_track_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_track_changed_s {
  bt_av_element_id_t id;
} bt_av_notification_track_changed_t;
```

Members

Members	Description
bt_av_element_id_t id;	< New track UID

Description

brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.

bt_av_notification_uids_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_uids_changed_s {
  bt_uint uid_counter;
} bt_av_notification_uids_changed_t;
```

Members

Members	Description
bt_uint uid_counter;	< UID counter

Description

brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.

bt_av_notification_volume_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_volume_changed_s {
  bt_byte volume;
} bt_av_notification_volume_changed_t;
```

Members

Members	Description
bt_byte volume;	< Volume

Description

brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.

bt_av_play_item_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_play_item_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_play_item_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of the request.

Description

brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.

bt_av_play_status_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_play_status_s {
  bt_av_response_t header;
  bt_ulong song_length;
  bt_ulong song_position;
  bt_byte play_status;
} bt_av_play_status_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_ulong song_length;	< Current track length
bt_ulong song_position;	< Current track position
bt_byte play_status;	< Playback status. This can be one of the following values: < li AVC_PLAY_STATUS_STOPPED < li AVC_PLAY_STATUS_PLAYING < li AVC_PLAY_STATUS_PAUSED < li AVC_PLAY_STATUS_FW_SEEK < li AVC_PLAY_STATUS_REV_SEEK < li AVC_PLAY_STATUS_ERROR

Description

brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.

bt_av_player_setting_current_values_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_current_values_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
  bt_byte* setting_value_id_list;
} bt_av_player_setting_current_values_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of player setting attribute ids to be returned from the target.
bt_byte* setting_id_list;	< List of player setting attribute ids to be returned from the target.
bt_byte* setting_value_id_list;	< List of current player setting attribute value ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.

bt_av_player_setting_values_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_values_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_value_id_list;
} bt_av_player_setting_values_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported player setting attribute value ids.
bt_byte* setting_value_id_list;	< List of supported player setting attribute value ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.

bt_av_player_setting_values_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_values_text_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_value_id_list;
  bt_av_player_text_t* setting_value_text_list;
} bt_av_player_setting_values_text_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of player setting attribute value ids for which displayable text is requested.
bt_byte* setting_value_id_list;	< List of player setting attribute value ids for which displayable text is requested.
bt_av_player_text_t* setting_value_text_list;	< List of player setting attribute values displayable text.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.

bt_av_player_settings_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_settings_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
} bt_av_player_settings_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported player setting attribute ids.

bt_byte* setting_id_list;	< List of supported player setting attribute ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.

bt_av_player_settings_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_settings_text_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
  bt_av_player_text_t* setting_text_list;
} bt_av_player_settings_text_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of player setting attribute ids for which displayable text is requested.
bt_byte* setting_id_list;	< List of player setting attribute ids for which displayable text is requested.
bt_av_player_text_t* setting_text_list;	< List of player setting attributes displayable text.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.

bt_av_player_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_text_s {
  bt_uint charset;
  bt_byte len;
  bt_byte* text;
} bt_av_player_text_t;
```

Description

This is type bt_av_player_text_t.

bt_av_register_notification_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_register_notification_t {
  bt_av_response_t header;
  bt_byte event_id;
  bt_ulong playback_pos;
} bt_av_register_notification_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte event_id;	< Event Id. This can be one of the following values: < li AVC_EVENT_PLAYBACK_STATUS_CHANGED < li AVC_EVENT_TRACK_CHANGED < li AVC_EVENT_TRACK_REACHED_END < li AVC_EVENT_TRACK_REACHED_START < li AVC_EVENT_PLAYBACK_POS_CHANGED < li AVC_EVENT_BATT_STATUS_CHANGED < li AVC_EVENT_SYSTEM_STATUS_CHANGED < li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED < li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED < li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED < li AVC_EVENT_ADDRESSED_PLAYER_CHANGED < li AVC_EVENT_UIDS_CHANGED < li AVC_EVENT_VOLUME_CHANGED
bt_ulong playback_pos;	< Playback position. Used only if c event_id is AVC_EVENT_PLAYBACK_POS_CHANGED

Description

brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.

bt_av_response_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_response_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte tran_id;
} bt_av_response_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel
bt_byte ctype;	< Response type. This can be one of the following values: < li AVC_RESPONSE_NOT_IMPLEMENTED < li AVC_RESPONSE_ACCEPTED < li AVC_RESPONSE_REJECTED < li AVC_RESPONSE_IN_TRANSITION < li AVC_RESPONSE_STABLE < li AVC_RESPONSE_IMPLEMENTED < li AVC_RESPONSE_CHANGED < li AVC_RESPONSE_INTERIM < li AVC_RESPONSE_TIMEOUT
bt_byte tran_id;	< Transaction Id.

Description

brief AV/C response header ingroup avrcp

details This structure is used to store fields present in every AV/C response.

bt_av_set_absolute_volume_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_set_absolute_volume_s {
  bt_av_response_t header;
  bt_byte volume;
} bt_av_set_absolute_volume_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.

bt_byte volume;	< Volume

Description

brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.

bt_av_set_addressed_player_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_set_addressed_player_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_set_addressed_player_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of changing the addressed player.

Description

brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.

bt_avrcp_channel_t Type

File

avrcp.h

C

```
typedef struct _bt_avrcp_channel_t bt_avrcp_channel_t;
```

Description

This is type bt_avrcp_channel_t.

bt_avrcp_device_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_device_s {
  bt_bdaddr_t address;
  bt_byte* name;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
} bt_avrcp_device_t;
```

Description

This is type bt_avrcp_device_t.

bt_avrcp_event_t Union

File

avrcp.h

C

```
typedef union _bt_avrcp_event_u {
  bt_avrcp_evt_channel_connected_t channel_connected;
  bt_avrcp_evt_channel_disconnected_t channel_disconnected;
  bt_avrcp_evt_connection_failed_t connection_failed;
  bt_avrcp_evt_panel_response_received_t panel_response_received;
  bt_avrcp_evt_search_completed_t device_search;
  bt_av_response_t response;
  bt_av_capability_company_id_t company_id;
  bt_av_capability_event_id_t supported_event_id;
  bt_av_player_settings_t player_settings;
  bt_av_player_setting_values_t player_setting_values;
  bt_av_player_setting_current_values_t player_setting_current_values;
  bt_av_player_settings_text_t player_settings_text;
  bt_av_player_setting_values_text_t player_setting_values_text;
  bt_av_element_attributes_t element_attributes;
  bt_av_play_status_t play_status;
  bt_av_set_absolute_volume_t absolute_volume;
  bt_av_set_addressed_player_t addressed_player;
  bt_av_play_item_t play_item_status;
  bt_av_add_to_now_playing_t add_to_now_playing_status;
  bt_av_notification_t notification;
  bt_avrcp_evt_register_events_completed_t register_events;
  bt_avrcp_evt_panel_command_received_t panel_command_received;
  bt_av_battery_status_of_ct_t battery_status_of_ct;
  bt_av_displayable_character_set_t displayable_character_set;
  bt_av_get_element_attributes_t get_element_attributes;
  bt_av_register_notification_t register_notification;
} bt_avrcp_event_t;
```

Members

Members	Description
bt_avrcp_evt_channel_connected_t channel_connected;	< Valid if event is AVRCP_EVT_CONTROL_CHANNEL_CONNECTED
bt_avrcp_evt_channel_disconnected_t channel_disconnected;	< Valid if event is AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED
bt_avrcp_evt_connection_failed_t connection_failed;	< Valid if event is AVRCP_EVT_CONTROL_CONNECTION_FAILED
bt_avrcp_evt_panel_response_received_t panel_response_received;	< Valid if event is AVRCP_EVT_PANEL_RESPONSE_RECEIVED
bt_avrcp_evt_search_completed_t device_search;	< Valid if event is AVRCP_EVT_SEARCH_COMPLETED
bt_av_capability_company_id_t company_id;	< Valid if event is AVRCP_EVT_COMPANY_ID_LIST_RECEIVED
bt_av_capability_event_id_t supported_event_id;	< Valid if event is AVRCP_EVT_EVENT_ID_LIST_RECEIVED
bt_av_player_settings_t player_settings;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED
bt_av_player_setting_values_t player_setting_values;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED
bt_av_player_setting_current_values_t player_setting_current_values;	< Valid if event is AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED
bt_av_player_settings_text_t player_settings_text;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED
bt_av_player_setting_values_text_t player_setting_values_text;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED
bt_av_element_attributes_t element_attributes;	< Valid if event is AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED
bt_av_play_status_t play_status;	< Valid if event is AVRCP_EVT_GET_PLAY_STATUS_RECEIVED
bt_av_set_absolute_volume_t absolute_volume;	< Valid if event is AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED
bt_av_set_addressed_player_t addressed_player;	< Valid if event is AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED
bt_av_play_item_t play_item_status;	< Valid if event is AVRCP_EVT_PLAY_ITEM_COMPLETED
bt_av_add_to_now_playing_t add_to_now_playing_status;	< Valid if event is AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED

bt_av_notification_t notification;	< Valid if event is on of the following: < li AVRCP_EVT_PLAYBACK_STATUS_CHANGED < li AVRCP_EVT_TRACK_CHANGED < li AVRCP_EVT_TRACK_REACHED_END < li AVRCP_EVT_TRACK_REACHED_START < li AVRCP_EVT_PLAYBACK_POS_CHANGED < li AVRCP_EVT_BATT_STATUS_CHANGED < li AVRCP_EVT_SYSTEM_STATUS_CHANGED < li AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED < li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED < li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED < li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED < li AVRCP_EVT_UIDS_CHANGED < li AVRCP_EVT_VOLUME_CHANGED
bt_avrcp_evt_register_events_completed_t register_events;	< Valid if event AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED
bt_avrcp_evt_panel_command_received_t panel_command_received;	< Valid if event is AVRCP_EVT_PANEL_COMMAND_RECEIVED
bt_av_battery_status_of_ct_t battery_status_of_ct;	< Valid if event is AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED
bt_av_displayable_character_set_t displayable_character_set;	< Valid if event is AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED
bt_av_get_element_attributes_t get_element_attributes;	< Valid if event is AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED
bt_av_register_notification_t register_notification;	< Valid if event is AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED

Description

brief Parameter to an application callback. ingroup avrcp

details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

bt_avrcp_evt_channel_connected_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_channel_connected_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_channel_connected_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.

bt_avrcp_evt_channel_disconnected_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_channel_disconnected_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_channel_disconnected_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.

bt_avrcp_evt_connection_failed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_connection_failed_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_connection_failed_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.

bt_avrcp_evt_panel_command_received_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_panel_command_received_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte button_status;
  bt_byte opcode;
  bt_byte* params;
  bt_byte params_len;
} bt_avrcp_evt_panel_command_received_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel
bt_byte ctype;	< Command type
bt_byte button_status;	< Button status
bt_byte opcode;	< Operation Id
bt_byte* params;	< Operation parameters
bt_byte params_len;	< Length of the operation parameters

Description

brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.

bt_avrcp_evt_panel_response_received_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_panel_response_received_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte button_status;
```

```
  bt_byte opcode;
  bt_byte* params;
  bt_byte params_len;
} bt_avrcp_evt_panel_response_received_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel
bt_byte ctype;	< Response type
bt_byte button_status;	< Button status
bt_byte opcode;	< Operation Id
bt_byte* params;	< Operation parameters
bt_byte params_len;	< Length of the operation parameters

Description

brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.

bt_avrcp_evt_register_events_completed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_register_events_completed_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_register_events_completed_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

This is type bt_avrcp_evt_register_events_completed_t.

bt_avrcp_evt_search_completed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_search_completed_s {
  bt_avrcp_device_t* devices;
  bt_byte count;
} bt_avrcp_evt_search_completed_t;
```

Members

Members	Description
bt_avrcp_device_t* devices;	< list of found devices
bt_byte count;	< the number of found devices

Description

brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.

bt_avrcp_find_callback_fp Type

File

avrcp.h

C

```
typedef void (* bt_avrcp_find_callback_fp)(bt_bool found, void* param);
```

Description

brief Find Controller/Target callback. ingroup avrcp

details This callback is called when search for Controller/Target has finished.

param found This can be one of the following values: li c TRUE if Controller/Target was found. li c FALSE otherwise. param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.

bt_avrcp_mgr_callback_fp Type

File

avrcp.h

C

```
typedef void (* bt_avrcp_mgr_callback_fp)(bt_avrcp_mgr_t* mgr, bt_byte evt, bt_avrcp_event_t* evt_param,
void* callback_param);
```

Description

brief AVRCP application callback. ingroup avrcp

details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated.

param mgr AVRCP manager.

param evt AVRCP event. The event can be one of the following values: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is generated when a local device failed to create a control channel between two AVRCP entities. @arg AVRCP_EVT_BROWSING_CHANNEL_CONNECTED This event is generated when a browsing channel between two AVRCP entities has been established. @arg AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED This event is generated when a browsing channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_BROWSING_CONNECTION_FAILED This event is generated when a local device failed to create a browsing channel between two AVRCP entities. @arg AVRCP_EVT_SEARCH_COMPLETED This event is generated when a local device completed searching for nearby targets.

@arg AVRCP_EVT_PANEL_RESPONSE_RECEIVED This event is generated when a local device received a response to a PASS THROUGH command. @arg AVRCP_EVT_COMPANY_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get company id" request. @arg AVRCP_EVT_EVENT_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get supported events" request. @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get supported player setting attributes" request. @arg AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values" request. @arg AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get current player setting attribute values" request. @arg AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED This event is generated when a local device received a response to a "set player setting attribute values" request. @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attributes displayable text" request. @arg AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values displayable text" request. @arg AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED This event is generated when a local device received a response to a "inform displayable character set" request. @arg AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED This event is generated when a local device received a response to a "inform battery status" request. @arg AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get media element attributes" request. @arg AVRCP_EVT_GET_PLAY_STATUS_RECEIVED This event is generated when a local device received a response to a "get play status" request. @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED This event is generated when a local device received a "play status changed" notification. @arg AVRCP_EVT_TRACK_CHANGED This event is generated when a local device received a "track changed changed" notification. @arg AVRCP_EVT_TRACK_REACHED_END This event is generated when a local device received a "track reached end" notification. @arg AVRCP_EVT_TRACK_REACHED_START This event is generated when a local device received a "track reached start" notification. @arg AVRCP_EVT_PLAYBACK_POS_CHANGED This event is generated when a local device received a "playback position changed" notification. @arg AVRCP_EVT_BATT_STATUS_CHANGED This event is generated when a local device received a "battery status changed" notification. @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED This event is generated when a local device received a "system status changed" notification.

@arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED This event is generated when a local device received a "player application setting changed" notification. @arg AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED This event is generated when a local device received a "now playing content changed" notification. @arg AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED This event is generated when a local device received a "available players changed" notification. @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED This event is generated when a local device received a "addressed player changed" notification. @arg AVRCP_EVT_UIDS_CHANGED This event is generated when a local device received a "UIDs changed" notification. @arg AVRCP_EVT_VOLUME_CHANGED This event is generated when a local device received a "volume changed" notification. @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED This event is generated when a local device received a response to a "set absolute volume" request. @arg AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED This event is generated when a local device received a response to a "set addressed player" request. @arg AVRCP_EVT_PLAY_ITEM_COMPLETED This event is generated when a local device received a response to a "play item" request. @arg AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED This event is generated when a local device received a response to a "add to now playing" request. @arg AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED This event is generated when a local device received a response to a "register notification" request.

@arg AVRCP_EVT_PANEL_COMMAND_RECEIVED This event is generated when a local device received a PASS THROUGH command. @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED This event is generated when a local device received a "set absolute volume" request. @arg AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED This event is generated when a local device received a "battery status of controller" command. @arg AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED This event is generated when a local device received a "displayable chracter set command" request. @arg AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED This event is generated when a local device received a "get element attributes" request.

param evt_param Event parameter. Which member of the bt_avrcp_event_t union is valid depends on the event: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED c bt_avrcp_evt_channel_connected_t channel_connected @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED c bt_avrcp_evt_channel_disconnected_t channel_disconnected @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED c bt_avrcp_evt_connection_failed_t connection_failed @arg AVRCP_EVT_PANEL_RESPONSE_RECEIVED c bt_avrcp_evt_panel_response_received_t panel_response_received @arg AVRCP_EVT_SEARCH_COMPLETED c bt_avrcp_evt_search_completed_t device_search @arg AVRCP_EVT_COMPANY_ID_LIST_RECEIVED c bt_av_capability_company_id_t company_id @arg AVRCP_EVT_EVENT_ID_LIST_RECEIVED c bt_av_capability_event_id_t supported_event_id @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED c bt_av_player_settings_t player_settings @arg AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED c bt_av_player_setting_values_t player_setting_values @arg AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED c bt_av_player_setting_current_values_t player_setting_current_values @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED c bt_av_player_settings_text_t player_settings_text @arg AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED c bt_av_player_setting_values_text_t player_setting_values_text @arg AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED c bt_av_element_attributes_t element_attributes @arg AVRCP_EVT_GET_PLAY_STATUS_RECEIVED c bt_av_play_status_t play_status @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED c bt_av_set_absolute_volume_t absolute_volume @arg AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED c bt_av_set_addressed_player_t addressed_player @arg AVRCP_EVT_PLAY_ITEM_COMPLETED c bt_av_play_item_t play_item_status @arg AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED c bt_av_add_to_now_playing_t add_to_now_playing_status @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_REACHED_END c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_REACHED_START c bt_av_notification_t notification @arg AVRCP_EVT_PLAYBACK_POS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_BATT_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_UIDS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_VOLUME_CHANGED c bt_av_notification_t notification

@arg AVRCP_EVT_PANEL_COMMAND_RECEIVED c bt_avrcp_evt_panel_command_received_t panel_command_received @arg AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED c bt_av_battery_status_of_ct_t battery_status_of_ct @arg AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED c bt_av_displayable_character_set_t displayable_character_set @arg AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED c bt_av_get_element_attributes_t get_element_attributes

param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_start.

bt_avrcp_mgr_t Type

File

avrcp.h

C

```
typedef struct _bt_avrcp_mgr_t bt_avrcp_mgr_t;
```

Description

This is type bt_avrcp_mgr_t.

__AVRCP_COMMAND_H Macro

File

avrcp_command.h

C

```
#define __AVRCP_COMMAND_H
```

Description

This is macro __AVRCP_COMMAND_H.

__AVRCP_H Macro

File

avrcp.h

C

```
#define __AVRCP_H
```

Description

This is macro __AVRCP_H.

__AVRCP_PRIVATE_H Macro

File

avrcp_private.h

C

```
#define __AVRCP_PRIVATE_H
```

Description

This is macro __AVRCP_PRIVATE_H.

AVC_BATTERY_STATUS_CRITICAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_CRITICAL 2
```

Description

This is macro AVC_BATTERY_STATUS_CRITICAL.

AVC_BATTERY_STATUS_EXTERNAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_EXTERNAL 3
```

Description

This is macro AVC_BATTERY_STATUS_EXTERNAL.

AVC_BATTERY_STATUS_FULL_CHARGE Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_FULL_CHARGE 4
```

Description

This is macro AVC_BATTERY_STATUS_FULL_CHARGE.

AVC_BATTERY_STATUS_NORMAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_NORMAL 0
```

Description

- addtogroup avrcp
- @{
- @name Battery status

AVC_BATTERY_STATUS_WARNING Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_WARNING 1
```

Description

This is macro AVC_BATTERY_STATUS_WARNING.

AVC_CAPABILITY_COMPANY_ID Macro

File

avrcp_command.h

C

```
#define AVC_CAPABILITY_COMPANY_ID 2
```

Description

This is macro AVC_CAPABILITY_COMPANY_ID.

AVC_CAPABILITY_EVENTS_SUPPORTED Macro

File

avrcp_command.h

C

```
#define AVC_CAPABILITY_EVENTS_SUPPORTED 3
```

Description

This is macro AVC_CAPABILITY_EVENTS_SUPPORTED.

AVC_CMD_GENERAL_POWER Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_POWER 0xB2
```

Description

This is macro AVC_CMD_GENERAL_POWER.

AVC_CMD_GENERAL_RESERVE Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_RESERVE 0x01
```

Description

This is macro AVC_CMD_GENERAL_RESERVE.

AVC_CMD_GENERAL_SUBUNIT_INFO Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_SUBUNIT_INFO 0x31
```

Description

This is macro AVC_CMD_GENERAL_SUBUNIT_INFO.

AVC_CMD_GENERAL_UNIT_INFO Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_UNIT_INFO 0x30
```

Description

This is macro AVC_CMD_GENERAL_UNIT_INFO.

AVC_CMD_GENERAL_VENDOR_DEPENDENT Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_VENDOR_DEPENDENT 0x00
```

Description

This is macro AVC_CMD_GENERAL_VENDOR_DEPENDENT.

AVC_CMD_GENERAL_VERSION Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_VERSION 0xB0
```

Description

This is macro AVC_CMD_GENERAL_VERSION.

AVC_CMD_PANEL_PASS_THROUGH Macro

File

avrcp_command.h

C

```
#define AVC_CMD_PANEL_PASS_THROUGH 0x7C
```

Description

This is macro AVC_CMD_PANEL_PASS_THROUGH.

AVC_CTYPE_CONTROL Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_CONTROL 0
```

Description

- addtogroup avrcp
- @{
- @name Command types

AVC_CTYPE_GENERAL_INQUORY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_GENERAL_INQUORY 4
```

Description

This is macro AVC_CTYPE_GENERAL_INQUORY.

AVC_CTYPE_NOTIFY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_NOTIFY 3
```

Description

This is macro AVC_CTYPE_NOTIFY.

AVC_CTYPE_SPECIFIC_IQUIRY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_SPECIFIC_IQUIRY 2
```

Description

This is macro AVC_CTYPE_SPECIFIC_IQUIRY.

AVC_CTYPE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_STATUS 1
```

Description

This is macro AVC_CTYPE_STATUS.

AVC_EVENT_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_ADDRESSED_PLAYER_CHANGED 0x0b ///< The Addressed Player has been changed, see 6.9.2.
```

Description

< The Addressed Player has been changed, see 6.9.2.

AVC_EVENT_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_AVAILABLE_PLAYERS_CHANGED 0x0a ///< The available players have changed, see 6.9
```

Description

< The available players have changed, see 6.9

AVC_EVENT_BATT_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_BATT_STATUS_CHANGED 0x06 ///< Change in battery status
```

Description

< Change in battery status

AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED 0x0400
```

Description

This is macro AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED.

AVC_EVENT_FLAG_ALL Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_ALL 0x1FFF
```

Description

This is macro AVC_EVENT_FLAG_ALL.

AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED 0x0200
```

Description

This is macro AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED.

AVC_EVENT_FLAG_BATT_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_BATT_STATUS_CHANGED 0x0020
```

Description

This is macro AVC_EVENT_FLAG_BATT_STATUS_CHANGED.

AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED 0x0100
```

Description

This is macro AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED.

AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED 0x0010
```

Description

This is macro AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED.

AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED 0x0001
```

Description

- addtogroup avrcp
- @{
- @name Notifications mask

AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED 0x0080
```

Description

This is macro AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED.

AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED 0x0040
```

Description

This is macro AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED.

AVC_EVENT_FLAG_TRACK_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_CHANGED 0x0002
```

Description

This is macro AVC_EVENT_FLAG_TRACK_CHANGED.

AVC_EVENT_FLAG_TRACK_REACHED_END Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_REACHED_END 0x0004
```

Description

This is macro AVC_EVENT_FLAG_TRACK_REACHED_END.

AVC_EVENT_FLAG_TRACK_REACHED_START Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_REACHED_START 0x0008
```

Description

This is macro AVC_EVENT_FLAG_TRACK_REACHED_START.

AVC_EVENT_FLAG_UIDS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_UIDS_CHANGED 0x0800
```

Description

This is macro AVC_EVENT_FLAG_UIDS_CHANGED.

AVC_EVENT_FLAG_VOLUME_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_VOLUME_CHANGED 0x1000
```

Description

This is macro AVC_EVENT_FLAG_VOLUME_CHANGED.

AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED 0x09 ///< The content of the Now Playing list has changed,
see 6.9.5.
```

Description

< The content of the Now Playing list has changed, see 6.9.5.

AVC_EVENT_PLAYBACK_POS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYBACK_POS_CHANGED 0x05 ///< Change in playback position. Returned after the specified playback notification change notification interval
```

Description

< Change in playback position. Returned after the specified playback notification change notification interval

AVC_EVENT_PLAYBACK_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYBACK_STATUS_CHANGED 0x01 ///< Change in playback status of the current track.
```

Description

< Change in playback status of the current track.

AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED 0x08 ///< Change in player application setting
```

Description

< Change in player application setting

AVC_EVENT_SYSTEM_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_SYSTEM_STATUS_CHANGED 0x07 ///< Change in system status
```

Description

< Change in system status

AVC_EVENT_TRACK_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_CHANGED 0x02 ///< Change of current track
```

Description

< Change of current track

AVC_EVENT_TRACK_REACHED_END Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_REACHED_END 0x03 ///< Reached end of a track
```

Description

< Reached end of a track

AVC_EVENT_TRACK_REACHED_START Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_REACHED_START 0x04 ///< Reached start of a track
```

Description

< Reached start of a track

AVC_EVENT_UIDS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_UIDS_CHANGED 0x0c ///< The UIDs have changed, see 6.10.3.3.
```

Description

< The UIDs have changed, see 6.10.3.3.

AVC_EVENT_VOLUME_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_VOLUME_CHANGED 0x0d ///< The volume has been changed locally on the TG, see 6.13.3
```

Description

< The volume has been changed locally on the TG, see 6.13.3

AVC_FLAG_BROWSING_CMD Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_BROWSING_CMD 2
```

Description

This is macro AVC_FLAG_BROWSING_CMD.

AVC_FLAG_PANEL_CLICK Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_PANEL_CLICK 1
```

Description

This is macro AVC_FLAG_PANEL_CLICK.

AVC_FLAG_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_RESPONSE 4
```

Description

This is macro AVC_FLAG_RESPONSE.

AVC_MAXEVENTS Macro

File

avrcp_command.h

C

```
#define AVC_MAXEVENTS 0x0d
```

Description

This is macro AVC_MAXEVENTS.

AVC_MEDIA_ATTR_FLAG_ALBUM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ALBUM 0x04
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ALBUM.

AVC_MEDIA_ATTR_FLAG_ALL Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ALL 0x7F
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ALL.

AVC_MEDIA_ATTR_FLAG_ARTIST Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ARTIST 0x02
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ARTIST.

AVC_MEDIA_ATTR_FLAG_GENRE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_GENRE 0x20
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_GENRE.

AVC_MEDIA_ATTR_FLAG_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_NUMBER 0x08
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_NUMBER.

AVC_MEDIA_ATTR_FLAG_PLAYING_TIME Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_PLAYING_TIME 0x40
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_PLAYING_TIME.

AVC_MEDIA_ATTR_FLAG_TITLE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_TITLE 0x01
```

Description

- addtogroup avrcp
- @{
- @name Media attribute bitmask

AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER 0x10
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER.

AVC_MEDIA_ATTR_ID_ALBUM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_ALBUM 3
```

Description

This is macro AVC_MEDIA_ATTR_ID_ALBUM.

AVC_MEDIA_ATTR_ID_ARTIST Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_ARTIST 2
```

Description

This is macro AVC_MEDIA_ATTR_ID_ARTIST.

AVC_MEDIA_ATTR_ID_GENRE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_GENRE 6
```

Description

This is macro AVC_MEDIA_ATTR_ID_GENRE.

AVC_MEDIA_ATTR_ID_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_NUMBER 4
```

Description

This is macro AVC_MEDIA_ATTR_ID_NUMBER.

AVC_MEDIA_ATTR_ID_PLAYING_TIME Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_PLAYING_TIME 7
```

Description

This is macro AVC_MEDIA_ATTR_ID_PLAYING_TIME.

AVC_MEDIA_ATTR_ID_TITLE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_TITLE 1
```

Description

- addtogroup avrcp
- @{
- @name Media attribute IDs

AVC_MEDIA_ATTR_ID_TOTAL_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_TOTAL_NUMBER 5
```

Description

This is macro AVC_MEDIA_ATTR_ID_TOTAL_NUMBER.

AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM 0x01 // Folder Item, Media Element Item  The virtual filesystem
containing the   Browsed
```

Description

Folder Item, Media Element Item The virtual filesystem containing the Browsed media content of the browsed player

AVC_NOW_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_NOW_PLAYING 0x03 // Media Element Item              The Now Playing list (or queue)
Addressed
```

Description

Media Element Item The Now Playing list (or queue) Addressed of the addressed player

AVC_PACKET_TYPE_CONTINUE Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_CONTINUE 2
```

Description

This is macro AVC_PACKET_TYPE_CONTINUE.

AVC_PACKET_TYPE_END Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_END 3
```

Description

This is macro AVC_PACKET_TYPE_END.

AVC_PACKET_TYPE_SINGLE Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_SINGLE 0
```

Description

This is macro AVC_PACKET_TYPE_SINGLE.

AVC_PACKET_TYPE_START Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_START 1
```

Description

This is macro AVC_PACKET_TYPE_START.

AVC_PANEL_BUTTON_PRESSED Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_BUTTON_PRESSED 0
```

Description

This is macro AVC_PANEL_BUTTON_PRESSED.

AVC_PANEL_BUTTON_RELEASED Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_BUTTON_RELEASED 1
```

Description

This is macro AVC_PANEL_BUTTON_RELEASED.

AVC_PANEL_OPID_0 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_0 0x20
```

Description

define AVC_PANEL_OPID_RESERVER 0x10 define AVC_PANEL_OPID_RESERVER 0x11 define AVC_PANEL_OPID_RESERVER 0x12 define AVC_PANEL_OPID_RESERVER 0x13 define AVC_PANEL_OPID_RESERVER 0x14 define AVC_PANEL_OPID_RESERVER 0x15 define AVC_PANEL_OPID_RESERVER 0x16 define AVC_PANEL_OPID_RESERVER 0x17 define AVC_PANEL_OPID_RESERVER 0x18 define AVC_PANEL_OPID_RESERVER 0x19 define AVC_PANEL_OPID_RESERVER 0x1A define AVC_PANEL_OPID_RESERVER 0x1B define AVC_PANEL_OPID_RESERVER 0x1C define AVC_PANEL_OPID_RESERVER 0x1D define AVC_PANEL_OPID_RESERVER 0x1E define AVC_PANEL_OPID_RESERVER 0x1F

AVC_PANEL_OPID_1 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_1 0x21
```

Description

This is macro AVC_PANEL_OPID_1.

AVC_PANEL_OPID_2 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_2 0x22
```

Description

This is macro AVC_PANEL_OPID_2.

AVC_PANEL_OPID_3 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_3 0x23
```

Description

This is macro AVC_PANEL_OPID_3.

AVC_PANEL_OPID_4 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_4 0x24
```

Description

This is macro AVC_PANEL_OPID_4.

AVC_PANEL_OPID_5 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_5 0x25
```

Description

This is macro AVC_PANEL_OPID_5.

AVC_PANEL_OPID_6 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_6 0x26
```

Description

This is macro AVC_PANEL_OPID_6.

AVC_PANEL_OPID_7 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_7 0x27
```

Description

This is macro AVC_PANEL_OPID_7.

AVC_PANEL_OPID_8 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_8 0x28
```

Description

This is macro AVC_PANEL_OPID_8.

AVC_PANEL_OPID_9 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_9 0x29
```

Description

This is macro AVC_PANEL_OPID_9.

AVC_PANEL_OPID_A Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_A 0x7A
```

Description

This is macro AVC_PANEL_OPID_A.

AVC_PANEL_OPID_ANGLE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ANGLE 0x50
```

Description

define AVC_PANEL_OPID_RESERVED 0x4E define AVC_PANEL_OPID_RESERVED 0x4F

AVC_PANEL_OPID_APPS_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_APPS_MENU 0x0F
```

Description

This is macro AVC_PANEL_OPID_APPS_MENU.

AVC_PANEL_OPID_B Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_B 0x7B
```

Description

This is macro AVC_PANEL_OPID_B.

AVC_PANEL_OPID_BACKWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_BACKWARD 0x4C
```

Description

This is macro AVC_PANEL_OPID_BACKWARD.

AVC_PANEL_OPID_C Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_C 0x7C
```

Description

This is macro AVC_PANEL_OPID_C.

AVC_PANEL_OPID_CHANNEL_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CHANNEL_DOWN 0x31
```

Description

This is macro AVC_PANEL_OPID_CHANNEL_DOWN.

AVC_PANEL_OPID_CHANNEL_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CHANNEL_UP 0x30
```

Description

define AVC_PANEL_OPID_RESRVED 0x2D define AVC_PANEL_OPID_RESRVED 0x2E define AVC_PANEL_OPID_RESRVED 0x2F

AVC_PANEL_OPID_CLEAR Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CLEAR 0x2C
```

Description

This is macro AVC_PANEL_OPID_CLEAR.

AVC_PANEL_OPID_CONTENTS_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CONTENTS_MENU 0x0B
```

Description

This is macro AVC_PANEL_OPID_CONTENTS_MENU.

AVC_PANEL_OPID_D Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_D 0x7D
```

Description

This is macro AVC_PANEL_OPID_D.

AVC_PANEL_OPID_DISPLAY_INFORMATION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DISPLAY_INFORMATION 0x35
```

Description

This is macro AVC_PANEL_OPID_DISPLAY_INFORMATION.

AVC_PANEL_OPID_DOT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DOT 0x2A
```

Description

This is macro AVC_PANEL_OPID_DOT.

AVC_PANEL_OPID_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DOWN 0x02
```

Description

This is macro AVC_PANEL_OPID_DOWN.

AVC_PANEL_OPID_EJECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_EJECT 0x4A
```

Description

This is macro AVC_PANEL_OPID_EJECT.

AVC_PANEL_OPID_ENTER Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ENTER 0x2B
```

Description

This is macro AVC_PANEL_OPID_ENTER.

AVC_PANEL_OPID_EXIT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_EXIT 0x0D
```

Description

This is macro AVC_PANEL_OPID_EXIT.

AVC_PANEL_OPID_F1 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F1 0x71
```

Description

define AVC_PANEL_OPID_RESERVED 0x6D define AVC_PANEL_OPID_RESERVED 0x6E define AVC_PANEL_OPID_RESERVED 0x6F define AVC_PANEL_OPID_RESERVED 0x70

AVC_PANEL_OPID_F2 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F2 0x72
```

Description

This is macro AVC_PANEL_OPID_F2.

AVC_PANEL_OPID_F3 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F3 0x73
```

Description

This is macro AVC_PANEL_OPID_F3.

AVC_PANEL_OPID_F4 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F4 0x74
```

Description

This is macro AVC_PANEL_OPID_F4.

AVC_PANEL_OPID_F5 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F5 0x75
```

Description

This is macro AVC_PANEL_OPID_F5.

AVC_PANEL_OPID_F6 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F6 0x76
```

Description

This is macro AVC_PANEL_OPID_F6.

AVC_PANEL_OPID_F7 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F7 0x77
```

Description

This is macro AVC_PANEL_OPID_F7.

AVC_PANEL_OPID_F8 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F8 0x78
```

Description

This is macro AVC_PANEL_OPID_F8.

AVC_PANEL_OPID_F9 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F9 0x79
```

Description

This is macro AVC_PANEL_OPID_F9.

AVC_PANEL_OPID_FAST_FORWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FAST_FORWARD 0x49
```

Description

This is macro AVC_PANEL_OPID_FAST_FORWARD.

AVC_PANEL_OPID_FAVORITE_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FAVORITE_MENU 0x0C
```

Description

This is macro AVC_PANEL_OPID_FAVORITE_MENU.

AVC_PANEL_OPID_FORWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FORWARD 0x4B
```

Description

This is macro AVC_PANEL_OPID_FORWARD.

AVC_PANEL_OPID_HELP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_HELP 0x36
```

Description

This is macro AVC_PANEL_OPID_HELP.

AVC_PANEL_OPID_INPUT_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_INPUT_SELECT 0x34
```

Description

This is macro AVC_PANEL_OPID_INPUT_SELECT.

AVC_PANEL_OPID_KEYBORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_KEYBORD_FUNCTION 0x6C
```

Description

This is macro AVC_PANEL_OPID_KEYBORD_FUNCTION.

AVC_PANEL_OPID_LEFT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT 0x03
```

Description

This is macro AVC_PANEL_OPID_LEFT.

AVC_PANEL_OPID_LEFT_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT_DOWN 0x08
```

Description

This is macro AVC_PANEL_OPID_LEFT_DOWN.

AVC_PANEL_OPID_LEFT_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT_UP 0x07
```

Description

This is macro AVC_PANEL_OPID_LEFT_UP.

AVC_PANEL_OPID_LINKED_CONTENT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LINKED_CONTENT 0x3F
```

Description

This is macro AVC_PANEL_OPID_LINKED_CONTENT.

AVC_PANEL_OPID_LIST Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LIST 0x4D
```

Description

This is macro AVC_PANEL_OPID_LIST.

AVC_PANEL_OPID_LIVE_TV Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LIVE_TV 0x39
```

Description

This is macro AVC_PANEL_OPID_LIVE_TV.

AVC_PANEL_OPID_LOCK Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LOCK 0x3B
```

Description

This is macro AVC_PANEL_OPID_LOCK.

AVC_PANEL_OPID_MUTE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_MUTE 0x43
```

Description

This is macro AVC_PANEL_OPID_MUTE.

AVC_PANEL_OPID_MUTE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_MUTE_FUNCTION 0x65
```

Description

This is macro AVC_PANEL_OPID_MUTE_FUNCTION.

AVC_PANEL_OPID_NEXT_DAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_NEXT_DAY 0x3D
```

Description

This is macro AVC_PANEL_OPID_NEXT_DAY.

AVC_PANEL_OPID_ON_DEMAND_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ON_DEMAND_MENU 0x0E
```

Description

This is macro AVC_PANEL_OPID_ON_DEMAND_MENU.

AVC_PANEL_OPID_PAGE_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAGE_DOWN 0x38
```

Description

This is macro AVC_PANEL_OPID_PAGE_DOWN.

AVC_PANEL_OPID_PAGE_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAGE_UP 0x37
```

Description

This is macro AVC_PANEL_OPID_PAGE_UP.

AVC_PANEL_OPID_PAUSE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE 0x46
```

Description

This is macro AVC_PANEL_OPID_PAUSE.

AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION 0x61
```

Description

This is macro AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION.

AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION 0x63
```

Description

This is macro AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION.

AVC_PANEL_OPID_PIP_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_DOWN 0x53
```

Description

This is macro AVC_PANEL_OPID_PIP_DOWN.

AVC_PANEL_OPID_PIP_MOVE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_MOVE 0x52
```

Description

This is macro AVC_PANEL_OPID_PIP_MOVE.

AVC_PANEL_OPID_PIP_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_UP 0x54
```

Description

This is macro AVC_PANEL_OPID_PIP_UP.

AVC_PANEL_OPID_PLAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PLAY 0x44
```

Description

This is macro AVC_PANEL_OPID_PLAY.

AVC_PANEL_OPID_PLAY_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PLAY_FUNCTION 0x60
```

Description

define AVC_PANEL_OPID_RESERVED 0x56 define AVC_PANEL_OPID_RESERVED 0x57 define AVC_PANEL_OPID_RESERVED 0x58 define AVC_PANEL_OPID_RESERVED 0x5A define AVC_PANEL_OPID_RESERVED 0x5B define AVC_PANEL_OPID_RESERVED 0x5C define AVC_PANEL_OPID_RESERVED 0x5D define AVC_PANEL_OPID_RESERVED 0x5E define AVC_PANEL_OPID_RESERVED 0x5F

AVC_PANEL_OPID_POWER_STATE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_POWER_STATE_FUNCTION 0x6B
```

Description

This is macro AVC_PANEL_OPID_POWER_STATE_FUNCTION.

AVC_PANEL_OPID_POWER_TOGGLE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_POWER_TOGGLE 0x40
```

Description

This is macro AVC_PANEL_OPID_POWER_TOGGLE.

AVC_PANEL_OPID_PREVIOUS_CHANNEL Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PREVIOUS_CHANNEL 0x32
```

Description

This is macro AVC_PANEL_OPID_PREVIOUS_CHANNEL.

AVC_PANEL_OPID_PREVIOUS_DAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PREVIOUS_DAY 0x3E
```

Description

This is macro AVC_PANEL_OPID_PREVIOUS_DAY.

AVC_PANEL_OPID_RECORD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RECORD 0x47
```

Description

This is macro AVC_PANEL_OPID_RECORD.

AVC_PANEL_OPID_RECORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RECORD_FUNCTION 0x62
```

Description

This is macro AVC_PANEL_OPID_RECORD_FUNCTION.

AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION 0x66
```

Description

This is macro AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION.

AVC_PANEL_OPID_REWIND Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_REWIND 0x48
```

Description

This is macro AVC_PANEL_OPID_REWIND.

AVC_PANEL_OPID_RF_BYPASS Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RF_BYPASS 0x55
```

Description

This is macro AVC_PANEL_OPID_RF_BYPASS.

AVC_PANEL_OPID_RIGHT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT 0x04
```

Description

This is macro AVC_PANEL_OPID_RIGHT.

AVC_PANEL_OPID_RIGHT_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT_DOWN 0x06
```

Description

This is macro AVC_PANEL_OPID_RIGHT_DOWN.

AVC_PANEL_OPID_RIGHT_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT_UP 0x05
```

Description

This is macro AVC_PANEL_OPID_RIGHT_UP.

AVC_PANEL_OPID_ROOT_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ROOT_MENU 0x09
```

Description

This is macro AVC_PANEL_OPID_ROOT_MENU.

AVC_PANEL_OPID_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT 0x00
```

Description

- addtogroup avrcp
- @{
- @name AV/C Panel PASS THROUGH operation IDs

AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION 0x6A
```

Description

This is macro AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION.

AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION 0x69
```

Description

This is macro AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION.

AVC_PANEL_OPID_SELECT_DISK_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_DISK_FUNCTION 0x68
```

Description

This is macro AVC_PANEL_OPID_SELECT_DISK_FUNCTION.

AVC_PANEL_OPID_SETUP_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SETUP_MENU 0x0A
```

Description

This is macro AVC_PANEL_OPID_SETUP_MENU.

AVC_PANEL_OPID_SKIP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SKIP 0x3C
```

Description

This is macro AVC_PANEL_OPID_SKIP.

AVC_PANEL_OPID_SOUND_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SOUND_SELECT 0x33
```

Description

This is macro AVC_PANEL_OPID_SOUND_SELECT.

AVC_PANEL_OPID_STOP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_STOP 0x45
```

Description

This is macro AVC_PANEL_OPID_STOP.

AVC_PANEL_OPID_STOP_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_STOP_FUNCTION 0x64
```

Description

This is macro AVC_PANEL_OPID_STOP_FUNCTION.

AVC_PANEL_OPID_SUBPICTURE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SUBPICTURE 0x51
```

Description

This is macro AVC_PANEL_OPID_SUBPICTURE.

AVC_PANEL_OPID_TUNE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_TUNE_FUNCTION 0x67
```

Description

This is macro AVC_PANEL_OPID_TUNE_FUNCTION.

AVC_PANEL_OPID_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_UP 0x01
```

Description

This is macro AVC_PANEL_OPID_UP.

AVC_PANEL_OPID_VENDOR_UNIQUE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VENDOR_UNIQUE 0x7E
```

Description

This is macro AVC_PANEL_OPID_VENDOR_UNIQUE.

AVC_PANEL_OPID_VOLUME_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VOLUME_DOWN 0x42
```

Description

This is macro AVC_PANEL_OPID_VOLUME_DOWN.

AVC_PANEL_OPID_VOLUME_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VOLUME_UP 0x41
```

Description

This is macro AVC_PANEL_OPID_VOLUME_UP.

AVC_PANEL_OPID_ZOOM Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ZOOM 0x3A
```

Description

This is macro AVC_PANEL_OPID_ZOOM.

AVC_PDUID_ABORT_CONTINUING_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_ABORT_CONTINUING_RESPONSE 0x41
```

Description

This is macro AVC_PDUID_ABORT_CONTINUING_RESPONSE.

AVC_PDUID_ADD_TO_NOW_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_ADD_TO_NOW_PLAYING 0x90
```

Description

This is macro AVC_PDUID_ADD_TO_NOW_PLAYING.

AVC_PDUID_CHANGE_PATH Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_CHANGE_PATH 0x72
```

Description

This is macro AVC_PDUID_CHANGE_PATH.

AVC_PDUID_GENERAL_REJECT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GENERAL_REJECT 0xa0
```

Description

This is macro AVC_PDUID_GENERAL_REJECT.

AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE 0x13
```

Description

This is macro AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE.

AVC_PDUID_GET_ELEMENT_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_ELEMENT_ATTRIBUTES 0x20
```

Description

This is macro AVC_PDUID_GET_ELEMENT_ATTRIBUTES.

AVC_PDUID_GET_FOLDER_ITEMS Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_FOLDER_ITEMS 0x71
```

Description

This is macro AVC_PDUID_GET_FOLDER_ITEMS.

AVC_PDUID_GET_ITEM_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_ITEM_ATTRIBUTES 0x73
```

Description

This is macro AVC_PDUID_GET_ITEM_ATTRIBUTES.

AVC_PDUID_GET_PLAY_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAY_STATUS 0x30
```

Description

This is macro AVC_PDUID_GET_PLAY_STATUS.

AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT 0x15
```

Description

This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT.

AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT 0x16
```

Description

This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT.

AVC_PDUID_GETCAPABILITIES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GETCAPABILITIES 0x10
```

Description

This is macro AVC_PDUID_GETCAPABILITIES.

AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT 0x18
```

Description

This is macro AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT.

AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET 0x17
```

Description

This is macro AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET.

AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES 0x11
```

Description

This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES.

AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES 0x12
```

Description

This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES.

AVC_PDUID_PLAY_ITEM Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_PLAY_ITEM 0x74
```

Description

This is macro AVC_PDUID_PLAY_ITEM.

AVC_PDUID_REGISTER_NOTIFICATION Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_REGISTER_NOTIFICATION 0x31
```

Description

This is macro AVC_PDUID_REGISTER_NOTIFICATION.

AVC_PDUID_REQUEST_CONTINUING_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_REQUEST_CONTINUING_RESPONSE 0x40
```

Description

This is macro AVC_PDUID_REQUEST_CONTINUING_RESPONSE.

AVC_PDUID_SEARCH Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SEARCH 0x80
```

Description

This is macro AVC_PDUID_SEARCH.

AVC_PDUID_SET_ABSOLUTE_VOLUME Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_ABSOLUTE_VOLUME 0x50
```

Description

This is macro AVC_PDUID_SET_ABSOLUTE_VOLUME.

AVC_PDUID_SET_ADDRESSED_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_ADDRESSED_PLAYER 0x60
```

Description

This is macro AVC_PDUID_SET_ADDRESSED_PLAYER.

AVC_PDUID_SET_BROWSED_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_BROWSED_PLAYER 0x70
```

Description

This is macro AVC_PDUID_SET_BROWSED_PLAYER.

AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE 0x14
```

Description

This is macro AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE.

AVC_PLAY_STATUS_ERROR Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_ERROR 0xFF
```

Description

This is macro AVC_PLAY_STATUS_ERROR.

AVC_PLAY_STATUS_FW_SEEK Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_FW_SEEK 3
```

Description

This is macro AVC_PLAY_STATUS_FW_SEEK.

AVC_PLAY_STATUS_PAUSED Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_PAUSED 2
```

Description

This is macro AVC_PLAY_STATUS_PAUSED.

AVC_PLAY_STATUS_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_PLAYING 1
```

Description

This is macro AVC_PLAY_STATUS_PLAYING.

AVC_PLAY_STATUS_REV_SEEK Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_REV_SEEK 4
```

Description

This is macro AVC_PLAY_STATUS_REV_SEEK.

AVC_PLAY_STATUS_STOPPED Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_STOPPED 0
```

Description

- addtogroup avrcp
- @{
- @name Play status

AVC_PLAYER_SETTING_EQUALIZER_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_OFF.

AVC_PLAYER_SETTING_EQUALIZER_ON Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_ON 2
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_ON.

AVC_PLAYER_SETTING_EQUALIZER_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_STATUS 1
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_STATUS.

AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS 3
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS.

AVC_PLAYER_SETTING_REPEAT_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_GROUP 4
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_GROUP.

AVC_PLAYER_SETTING_REPEAT_MODE_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_MODE_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_MODE_OFF.

AVC_PLAYER_SETTING_REPEAT_MODE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_MODE_STATUS 2
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_MODE_STATUS.

AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK 2
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK.

AVC_PLAYER_SETTING_SCAN_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_ALL_TRACKS 2
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_ALL_TRACKS.

AVC_PLAYER_SETTING_SCAN_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_GROUP 3
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_GROUP.

AVC_PLAYER_SETTING_SCAN_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_OFF.

AVC_PLAYER_SETTING_SCAN_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_STATUS 4
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_STATUS.

AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS 2
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS.

AVC_PLAYER_SETTING_SHUFFLE_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_GROUP 3
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_GROUP.

AVC_PLAYER_SETTING_SHUFFLE_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_OFF.

AVC_PLAYER_SETTING_SHUFFLE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_STATUS 3
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_STATUS.

AVC_RESPONSE_ACCEPTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_ACCEPTED 0x9
```

Description

This is macro AVC_RESPONSE_ACCEPTED.

AVC_RESPONSE_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_CHANGED 0xD
```

Description

This is macro AVC_RESPONSE_CHANGED.

AVC_RESPONSE_IMPLEMENTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_IMPLEMENTED 0xC
```

Description

This is macro AVC_RESPONSE_IMPLEMENTED.

AVC_RESPONSE_IN_TRANSITION Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_IN_TRANSITION 0xB
```

Description

This is macro AVC_RESPONSE_IN_TRANSITION.

AVC_RESPONSE_INTERIM Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_INTERIM 0xF
```

Description

This is macro AVC_RESPONSE_INTERIM.

AVC_RESPONSE_NOT_IMPLEMENTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_NOT_IMPLEMENTED 0x8
```

Description

- addtogroup avrcp
- @{
- @name Response types

AVC_RESPONSE_REJECTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_REJECTED 0xA
```

Description

This is macro AVC_RESPONSE_REJECTED.

AVC_RESPONSE_STABLE Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_STABLE 0xC
```

Description

This is macro AVC_RESPONSE_STABLE.

AVC_RESPONSE_TIMEOUT Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_TIMEOUT 0xF0
```

Description

This is macro AVC_RESPONSE_TIMEOUT.

AVC_SCOPE_MEDIA_PLAYER_LIST Macro

File

avrcp_command.h

C

```
#define AVC_SCOPE_MEDIA_PLAYER_LIST 0x00 // Media Player Item                Contains all available media
players    None
```

Description

Media Player Item Contains all available media players None

AVC_SEARCH Macro

File

avrcp_command.h

C

```
#define AVC_SEARCH 0x02 // Media Element Item              The results of a search operation       Browsed
```

Description

Media Element Item The results of a search operation Browsed on the browsed player

AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE 0x5
```

Description

This is macro AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE.

AVC_SUBUNIT_ID_IGNORE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_ID_IGNORE 0x7
```

Description

This is macro AVC_SUBUNIT_ID_IGNORE.

AVC_SUBUNIT_TYPE_AUDIO Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_AUDIO 0x01
```

Description

This is macro AVC_SUBUNIT_TYPE_AUDIO.

AVC_SUBUNIT_TYPE_BULLETIN_BOARD Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_BULLETIN_BOARD 0x0A
```

Description

This is macro AVC_SUBUNIT_TYPE_BULLETIN_BOARD.

AVC_SUBUNIT_TYPE_CA Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CA 0x06
```

Description

This is macro AVC_SUBUNIT_TYPE_CA.

AVC_SUBUNIT_TYPE_CAMERA Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CAMERA 0x07
```

Description

This is macro AVC_SUBUNIT_TYPE_CAMERA.

AVC_SUBUNIT_TYPE_CAMERA_STORAGE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CAMERA_STORAGE 0x0B
```

Description

This is macro AVC_SUBUNIT_TYPE_CAMERA_STORAGE.

AVC_SUBUNIT_TYPE_DISC Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_DISC 0x03
```

Description

This is macro AVC_SUBUNIT_TYPE_DISC.

AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE 0x1E
```

Description

This is macro AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE.

AVC_SUBUNIT_TYPE_MONITOR Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_MONITOR 0x00
```

Description

- addtogroup avrcp
- @{
- @name Subunit types

AVC_SUBUNIT_TYPE_PANEL Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_PANEL 0x09
```

Description

This is macro AVC_SUBUNIT_TYPE_PANEL.

AVC_SUBUNIT_TYPE_PRINTER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_PRINTER 0x02
```

Description

This is macro AVC_SUBUNIT_TYPE_PRINTER.

AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER 0x04
```

Description

This is macro AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER.

AVC_SUBUNIT_TYPE_TUNER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_TUNER 0x05
```

Description

This is macro AVC_SUBUNIT_TYPE_TUNER.

AVC_SUBUNIT_TYPE_UNIT Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_UNIT 0x1F
```

Description

This is macro AVC_SUBUNIT_TYPE_UNIT.

AVC_SUBUNIT_TYPE_VENDOR_UNIQUE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_VENDOR_UNIQUE 0x1C
```

Description

This is macro AVC_SUBUNIT_TYPE_VENDOR_UNIQUE.

AVC_VOLUME_MAX Macro

File

avrcp_command.h

C

```
#define AVC_VOLUME_MAX 0x7F // 100%
```

Description

100

AVC_VOLUME_MIN Macro

File

avrcp_command.h

C

```
#define AVC_VOLUME_MIN 0 // 0%
```

Description

0

AVRCP_BTSIG_COMPANY_ID Macro

File

avrcp.h

C

```
#define AVRCP_BTSIG_COMPANY_ID 0x001958
```

Description

This is macro AVRCP_BTSIG_COMPANY_ID.

AVRCP_CHANNEL_FLAG_LISTENING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_LISTENING 1
```

Description

This is macro AVRCP_CHANNEL_FLAG_LISTENING.

AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED 8
```

Description

This is macro AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED.

AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS 4
```

Description

This is macro AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS.

AVRCP_CHANNEL_FLAG_SENDING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_SENDING 2
```

Description

This is macro AVRCP_CHANNEL_FLAG_SENDING.

AVRCP_CHANNEL_STATE_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_CONNECTED 3
```

Description

This is macro AVRCP_CHANNEL_STATE_CONNECTED.

AVRCP_CHANNEL_STATE_CONNECTING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVRCP_CHANNEL_STATE_CONNECTING.

AVRCP_CHANNEL_STATE_DISCONNECTING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_DISCONNECTING 4
```

Description

This is macro AVRCP_CHANNEL_STATE_DISCONNECTING.

AVRCP_CHANNEL_STATE_FREE Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_FREE 0
```

Description

This is macro AVRCP_CHANNEL_STATE_FREE.

AVRCP_CHANNEL_STATE_IDLE Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_IDLE 1
```

Description

This is macro AVRCP_CHANNEL_STATE_IDLE.

AVRCP_COMMAND_TYPE_BROWSING Macro

File

avrcp_command.h

C

```
#define AVRCP_COMMAND_TYPE_BROWSING 1
```

Description

This is macro AVRCP_COMMAND_TYPE_BROWSING.

AVRCP_COMMAND_TYPE_CONTROL Macro

File

avrcp_command.h

C

```
#define AVRCP_COMMAND_TYPE_CONTROL 0
```

Description

This is macro AVRCP_COMMAND_TYPE_CONTROL.

AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED 79    ///< This event is generated when a local device
received a response to a "add to now playing" request.
```

Description

< This event is generated when a local device received a response to a "add to now playing" request.

AVRCP_EVT_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ADDRESSED_PLAYER_CHANGED 73    ///< This event is generated when a local device received
a "addressed player changed" notification.
```

Description

< This event is generated when a local device received a "addressed player changed" notification.

AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED 72    ///< This event is generated when a local device received
a "available players changed" notification.
```

Description

< This event is generated when a local device received a "available players changed" notification.

AVRCP_EVT_BATT_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BATT_STATUS_CHANGED 68    ///< This event is generated when a local device received a
"battery status changed" notification.
```

Description

< This event is generated when a local device received a "battery status changed" notification.

AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED 83    ///< This event is generated when a local device
received a "battery status of controller" command.
```

Description

< This event is generated when a local device received a "battery status of controller" command.

AVRCP_EVT_BROWSING_CHANNEL_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CHANNEL_CONNECTED 4    ///< This event is generated when a browsing channel
between two AVRCP entities has been established.
```

Description

< This event is generated when a browsing channel between two AVRCP entities has been established.

AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED 5    ///< This event is generated when a browsing channel
between two AVRCP entities has been terminated.
```

Description

< This event is generated when a browsing channel between two AVRCP entities has been terminated.

AVRCP_EVT_BROWSING_CONNECTION_FAILED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CONNECTION_FAILED 6    ///< This event is generated when a local device failed
to create a browsing channel between two AVRCP entities.
```

Description

< This event is generated when a local device failed to create a browsing channel between two AVRCP entities.

AVRCP_EVT_COMPANY_ID_LIST_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_COMPANY_ID_LIST_RECEIVED 51    ///< This event is generated when a local device received
a response to a "get company id" request.
```

Description

< This event is generated when a local device received a response to a "get company id" request.

AVRCP_EVT_CONTROL_CHANNEL_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CHANNEL_CONNECTED 1    ///< This event is generated when a control channel
between two AVRCP entities has been established.
```

Description

< This event is generated when a control channel between two AVRCP entities has been established.

AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED 2    ///< This event is generated when a control channel
between two AVRCP entities has been terminated.
```

Description

< This event is generated when a control channel between two AVRCP entities has been terminated.

AVRCP_EVT_CONTROL_CONNECTION_FAILED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CONNECTION_FAILED 3    ///< This event is generated when a local device failed to
create a control channel between two AVRCP entities.
```

Description

< This event is generated when a local device failed to create a control channel between two AVRCP entities.

AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED 84    ///< This event is generated when a local device
received a "displayable chracter set command" request.
```

Description

< This event is generated when a local device received a "displayable chracter set command" request.

AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED 85    ///< This event is generated when a local device
received a "get element attributes" request.
```

Description

< This event is generated when a local device received a "get element attributes" request.

AVRCP_EVT_EVENT_ID_LIST_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_EVENT_ID_LIST_RECEIVED 52    ///< This event is generated when a local device received a
response to a "get supported events" request.
```

Description

< This event is generated when a local device received a response to a "get supported events" request.

AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED 61    ///< This event is generated when a local device received a response to a "get media element attributes" request.
```

Description

< This event is generated when a local device received a response to a "get media element attributes" request.

AVRCP_EVT_GET_PLAY_STATUS_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_GET_PLAY_STATUS_RECEIVED 62    ///< This event is generated when a local device received a response to a "get play status" request.
```

Description

< This event is generated when a local device received a response to a "get play status" request.

AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED 60    ///< This event is generated when a local device received a response to a "inform battery status" request.
```

Description

< This event is generated when a local device received a response to a "inform battery status" request.

AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED 59    ///< This event is generated when a local device received a response to a "inform displayable character set" request.
```

Description

< This event is generated when a local device received a response to a "inform displayable character set" request.

AVRCP_EVT_NOTHING Macro

File

avrcp.h

C

```
#define AVRCP_EVT_NOTHING 0
```

Description

addtogroup avrcp @{

@name Events

details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED 71    ///< This event is generated when a local device
received a "now playing content changed" notification.
```

Description

< This event is generated when a local device received a "now playing content changed" notification.

AVRCP_EVT_PANEL_COMMAND_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PANEL_COMMAND_RECEIVED 81    ///< This event is generated when a local device received a
PASS THROUGH command.
```

Description

< This event is generated when a local device received a PASS THROUGH command.

AVRCP_EVT_PANEL_RESPONSE_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PANEL_RESPONSE_RECEIVED 50    ///< This event is generated when a local device received a
response to a PASS THROUGH command.
```

Description

< This event is generated when a local device received a response to a PASS THROUGH command.

AVRCP_EVT_PLAY_ITEM_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAY_ITEM_COMPLETED 78    ///< This event is generated when a local device received a
response to a "play item" request.
```

Description

< This event is generated when a local device received a response to a "play item" request.

AVRCP_EVT_PLAYBACK_POS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYBACK_POS_CHANGED 67    ///< This event is generated when a local device received a
"playback position changed" notification.
```

Description

< This event is generated when a local device received a "playback position changed" notification.

AVRCP_EVT_PLAYBACK_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYBACK_STATUS_CHANGED 63    ///< This event is generated when a local device received a "play status changed" notification.
```

Description

< This event is generated when a local device received a "play status changed" notification.

AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED 70    ///< This event is generated when a local device received a "player application setting changed" notification.
```

Description

< This event is generated when a local device received a "player application setting changed" notification.

AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED 55    ///< This event is generated when a local device received a response to a "get current player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "get current player setting attribute values" request.

AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED 53    ///< This event is generated when a local device received a response to a "get supported player setting attributes" request.
```

Description

< This event is generated when a local device received a response to a "get supported player setting attributes" request.

AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED 57    ///< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.

AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED 54    ///< This event is generated when a local device received a response to a "get player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attribute values" request.

AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED 58    ///< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.

AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED 86    ///< This event is generated when a local device received a "register notification" request.
```

Description

< This event is generated when a local device received a "register notification" request.

AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED 80    ///< This event is generated when a local device received a response to a "register notification" request.
```

Description

< This event is generated when a local device received a response to a "register notification" request.

AVRCP_EVT_SEARCH_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SEARCH_COMPLETED 7    ///< This event is generated when a local device completed searching for nearby targets.
```

Description

< This event is generated when a local device completed searching for nearby targets.

AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED 76    ///< This event is generated when a local device received a response to a "set absolute volume" request.
```

Description

< This event is generated when a local device received a response to a "set absolute volume" request.

AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED 82    ///< This event is generated when a local device received a "set absolute volume" request.
```

Description

< This event is generated when a local device received a "set absolute volume" request.

AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED 77    ///< This event is generated when a local device received a response to a "set addressed player" request.
```

Description

< This event is generated when a local device received a response to a "set addressed player" request.

AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED 56    ///< This event is generated when a local device received a response to a "set player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "set player setting attribute values" request.

AVRCP_EVT_SYSTEM_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SYSTEM_STATUS_CHANGED 69    ///< This event is generated when a local device received a "system status changed" notification.
```

Description

< This event is generated when a local device received a "system status changed" notification.

AVRCP_EVT_TRACK_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_CHANGED 64    ///< This event is generated when a local device received a "track changed changed" notification.
```

Description

< This event is generated when a local device received a "track changed changed" notification.

AVRCP_EVT_TRACK_REACHED_END Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_REACHED_END 65    ///< This event is generated when a local device received a "track reached end" notification.
```

Description

< This event is generated when a local device received a "track reached end" notification.

AVRCP_EVT_TRACK_REACHED_START Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_REACHED_START 66    ///< This event is generated when a local device received a "track reached start" notification.
```

Description

< This event is generated when a local device received a "track reached start" notification.

AVRCP_EVT_UIDS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_UIDS_CHANGED 74    ///< This event is generated when a local device received a "UIDs changed" notification.
```

Description

< This event is generated when a local device received a "UIDs changed" notification.

AVRCP_EVT_VOLUME_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_VOLUME_CHANGED 75    ///< This event is generated when a local device received a "volume changed" notification.
```

Description

< This event is generated when a local device received a "volume changed" notification.

AVRCP_MANAGER_FLAG_SEARCHING Macro

File

avrcp.h

C

```
#define AVRCP_MANAGER_FLAG_SEARCHING 1
```

Description

This is macro AVRCP_MANAGER_FLAG_SEARCHING.

AVRCP_MANAGER_STATE_IDLE Macro

File

avrcp.h

C

```
#define AVRCP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVRCP_MANAGER_STATE_IDLE.

AVRCP_MAX_ELEMENT_ATTRIBUTES Macro

File

avrcp_private.h

C

```
#define AVRCP_MAX_ELEMENT_ATTRIBUTES 7
```

Description

This is macro AVRCP_MAX_ELEMENT_ATTRIBUTES.

bt_avrcp_0_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_click.

bt_avrcp_0_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_press.

bt_avrcp_1_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_click.

bt_avrcp_1_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_press.

bt_avrcp_2_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_click.

bt_avrcp_2_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_press.

bt_avrcp_3_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_click.

bt_avrcp_3_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_press.

bt_avrcp_4_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_click.

bt_avrcp_4_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_press.

bt_avrcp_5_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_click.

bt_avrcp_5_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_press.

bt_avrcp_6_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_click.

bt_avrcp_6_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_press.

bt_avrcp_7_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_click.

bt_avrcp_7_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_press.

bt_avrcp_8_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_click.

bt_avrcp_8_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_press.

bt_avrcp_9_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_click.

bt_avrcp_9_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_press.

bt_avrcp_angle_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_click.

bt_avrcp_angle_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_press.

bt_avrcp_backward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_click.

bt_avrcp_backward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_press.

bt_avrcp_channel_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_click.

bt_avrcp_channel_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_press.

bt_avrcp_channel_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_click.

bt_avrcp_channel_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_press.

bt_avrcp_clear_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_click.

bt_avrcp_clear_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_press.

bt_avrcp_content_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_content_menu_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_content_menu_click.

bt_avrcp_contents_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_contents_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_contents_menu_press.

bt_avrcp_display_info_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_click.

bt_avrcp_display_info_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_press.

bt_avrcp_dot_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_click.

bt_avrcp_dot_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_press.

bt_avrcp_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_click.

bt_avrcp_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_press.

bt_avrcp_eject_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_click.

bt_avrcp_eject_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_press.

bt_avrcp_enter_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_click.

bt_avrcp_enter_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_press.

bt_avrcp_exit_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_click.

bt_avrcp_exit_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_press.

bt_avrcp_f1_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_click.

bt_avrcp_f1_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_press.

bt_avrcp_f2_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_click.

bt_avrcp_f2_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_press.

bt_avrcp_f3_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_click.

bt_avrcp_f3_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_press.

bt_avrcp_f4_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_click.

bt_avrcp_f4_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_press.

bt_avrcp_f5_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_click.

bt_avrcp_f5_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_press.

bt_avrcp_f6_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_click.

bt_avrcp_f6_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_press.

bt_avrcp_f7_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_click.

bt_avrcp_f7_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_press.

bt_avrcp_f8_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_click.

bt_avrcp_f8_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_press.

bt_avrcp_f9_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_click.

bt_avrcp_f9_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_press.

bt_avrcp_fast_forward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_click.

bt_avrcp_fast_forward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_press.

bt_avrcp_favorite_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorite_menu_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_FAVORITE_MENU)
```

Description

This is macro bt_avrcp_favorite_menu_click.

bt_avrcp_favorive_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorive_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_FAVORIVE_MENU)
```

Description

This is macro bt_avrcp_favorive_menu_press.

bt_avrcp_forward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_click.

bt_avrcp_forward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_press.

bt_avrcp_help_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_click.

bt_avrcp_help_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_press.

bt_avrcp_input_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_click.

bt_avrcp_input_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_press.

bt_avrcp_left_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_click.

bt_avrcp_left_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_click.

bt_avrcp_left_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_press.

bt_avrcp_left_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_press.

bt_avrcp_left_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_click.

bt_avrcp_left_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_press.

bt_avrcp_mute_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_click.

bt_avrcp_mute_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_press.

bt_avrcp_page_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_click.

bt_avrcp_page_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_press.

bt_avrcp_page_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_click.

bt_avrcp_page_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_press.

bt_avrcp_pause_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_click.

bt_avrcp_pause_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_press.

bt_avrcp_play_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_click.

bt_avrcp_play_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_press.

bt_avrcp_power_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_click.

bt_avrcp_power_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_press.

bt_avrcp_previous_channel_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_click.

bt_avrcp_previous_channel_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_press.

bt_avrcp_record_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_click.

bt_avrcp_record_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_press.

bt_avrcp_rewind_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_click.

bt_avrcp_rewind_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_press.

bt_avrcp_right_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_click.

bt_avrcp_right_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_click.

bt_avrcp_right_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_press.

bt_avrcp_right_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_press.

bt_avrcp_right_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_click.

bt_avrcp_right_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_press.

bt_avrcp_root_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_click.

bt_avrcp_root_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_press.

bt_avrcp_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SELECT)
```

Description

This is macro bt_avrcp_select_click.

bt_avrcp_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_SELECT)
```

Description

Panel operations

bt_avrcp_setup_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_click.

bt_avrcp_setup_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_press.

bt_avrcp_sound_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_click.

bt_avrcp_sound_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_press.

bt_avrcp_stop_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_click.

bt_avrcp_stop_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_press.

bt_avrcp_subpicture_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_click.

bt_avrcp_subpicture_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_press.

bt_avrcp_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_click.

bt_avrcp_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_press.

bt_avrcp_volume_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_click.

bt_avrcp_volume_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_press.

bt_avrcp_volume_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_click.

bt_avrcp_volume_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_press.

bt_avrcp_0_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_release.

bt_avrcp_1_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_release.

bt_avrcp_2_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_release.

bt_avrcp_3_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_release.

bt_avrcp_4_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_release.

bt_avrcp_5_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_release.

bt_avrcp_6_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_release.

bt_avrcp_7_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_release.

bt_avrcp_8_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_release.

bt_avrcp_9_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_release.

bt_avrcp_angle_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_release.

bt_avrcp_backward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_release.

bt_avrcp_channel_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_release.

bt_avrcp_channel_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_release.

bt_avrcp_clear_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_release.

bt_avrcp_content_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_content_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_content_menu_release.

bt_avrcp_display_info_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_release.

bt_avrcp_dot_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_release.

bt_avrcp_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_release.

bt_avrcp_eject_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_release.

bt_avrcp_enter_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_release.

bt_avrcp_exit_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_release.

bt_avrcp_f1_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_release.

bt_avrcp_f2_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_release.

bt_avrcp_f3_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_release.

bt_avrcp_f4_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_release.

bt_avrcp_f5_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_release.

bt_avrcp_f6_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_release.

bt_avrcp_f7_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_release.

bt_avrcp_f8_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_release.

bt_avrcp_f9_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_release.

bt_avrcp_fast_forward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_release.

bt_avrcp_favorite_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorite_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FAVORIVE_MENU)
```

Description

This is macro bt_avrcp_favorite_menu_release.

bt_avrcp_forward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_release.

bt_avrcp_help_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_release.

bt_avrcp_input_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_release.

bt_avrcp_left_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_release.

bt_avrcp_left_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_release.

bt_avrcp_left_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_release.

bt_avrcp_mute_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_release.

bt_avrcp_page_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_release.

bt_avrcp_page_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_release.

bt_avrcp_pause_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_release.

bt_avrcp_play_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_release.

bt_avrcp_power_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_release.

bt_avrcp_previous_channel_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_release.

bt_avrcp_record_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_release.

bt_avrcp_rewind_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_release.

bt_avrcp_right_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_release.

bt_avrcp_right_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_release.

bt_avrcp_right_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_release.

bt_avrcp_root_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_release.

bt_avrcp_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_SELECT)
```

Description

This is macro bt_avrcp_select_release.

bt_avrcp_setup_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_release.

bt_avrcp_sound_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_release.

bt_avrcp_stop_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_release.

bt_avrcp_subpicture_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_release.

bt_avrcp_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_release.

bt_avrcp_volume_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_release.

bt_avrcp_volume_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_release.

__AVRCP_CONFIG_EVENT_HANDLERS_H Macro

File

avrcp_config_event_handlers.h

C

```
#define __AVRCP_CONFIG_EVENT_HANDLERS_H
```

Description

This is macro __AVRCP_CONFIG_EVENT_HANDLERS_H.

AVRCP_COMMAND_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_COMMAND_HANDLER NULL
```

Description

This is macro AVRCP_COMMAND_HANDLER.

AVRCP_COMMAND_SENT_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_COMMAND_SENT_HANDLER NULL
```

Description

This is macro AVRCP_COMMAND_SENT_HANDLER.

AVRCP_RESPONSE_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_RESPONSE_HANDLER NULL
```

Description

This is macro AVRCP_RESPONSE_HANDLER.

AVRCP_RESPONSE_SENT_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_RESPONSE_SENT_HANDLER NULL
```

Description

This is macro AVRCP_RESPONSE_SENT_HANDLER.

GAP Functions

bt_gap_find_devices Function

File

gap.h

C

```
bt_bool bt_gap_find_devices(bt_device_t* devices, bt_byte max_devices, bt_byte length, bt_byte mode,
bt_find_devices_callback_fp callback, void* callback_param);
```

Description

This is function bt_gap_find_devices.

GAP Data Types and Constants

bt_device_t Structure

File

gap.h

C

```
typedef struct _bt_device_t {
  bt_bdaddr_t bdaddr;
  bt_byte* name;
  bt_byte max_name_len;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
  bt_byte* eir;
  bt_byte eir_len;
  bt_byte max_eir_len;
} bt_device_t;
```

Description

This is type bt_device_t.

bt_find_devices_callback_fp Type

File

gap.h

C

```
typedef void (* bt_find_devices_callback_fp)(bt_byte count, void* callback_param);
```

Description

This is type bt_find_devices_callback_fp.

HCI Functions

bt_hci_add_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_add_param_bdaddr(bt_hci_command_p pcmd, const bt_bdaddr_t* pbdaddr);
```

Description

brief Add BD address parameter to an HCI command ingroup hci

bt_hci_add_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_add_param_byte(bt_hci_command_p pcmd, bt_byte value);
```

Description

brief Add byte parameter to an HCI command ingroup hci

bt_hci_add_param_cod Function

File

hci.h

C

```
bt_bool bt_hci_add_param_cod(bt_hci_command_p pcmd, bt_long value);
```

Description

brief Add class of device parameter to an HCI command ingroup hci

bt_hci_add_param_int Function

File

hci.h

C

```
bt_bool bt_hci_add_param_int(bt_hci_command_p pcmd, bt_int value);
```

Description

brief Add int parameter to an HCI command ingroup hci

bt_hci_add_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_add_param_linkkey(bt_hci_command_p pcmd, const bt_linkkey_t* linkkey);
```

Description

brief Add link key parameter to an HCI command ingroup hci

bt_hci_add_param_long Function

File

hci.h

C

```
bt_bool bt_hci_add_param_long(bt_hci_command_p pcmd, bt_long value);
```

Description

brief Add long parameter to an HCI command ingroup hci

bt_hci_add_param_string Function

File

hci.h

C

```
bt_bool bt_hci_add_param_string(bt_hci_command_p pcmd, const char * ps, bt_int len);
```

Description

brief Add string parameter to an HCI command ingroup hci

bt_hci_alloc_canned_command Function

File

hci.h

C

```
bt_hci_command_p bt_hci_alloc_canned_command(const bt_byte* canned_command, bt_hci_cmd_callback_fp
callback);
```

Description

brief Allocate and initialize an HCI command structure for a canned (pre-formatted) command. ingroup hci

bt_hci_alloc_command Function

File

hci.h

C

```
bt_hci_command_p bt_hci_alloc_command(bt_int opcode, bt_hci_cmd_callback_fp callback);
```

Description

brief Allocate and initialize an HCI command structure. ingroup hci

bt_hci_alloc_data_buffer Function

File

hci_data_buffer.h

C

```
bt_hci_data_p bt_hci_alloc_data_buffer();
```

Description

This is function bt_hci_alloc_data_buffer.

bt_hci_allocate_write_eir_command Function

File

hci_eir.h

C

```
bt_hci_command_p bt_hci_allocate_write_eir_command(bt_byte fec_required);
```

Description

This is function bt_hci_allocate_write_eir_command.

bt_hci_authenticate_ex Function

File

hci.h

C

```
bt_bool bt_hci_authenticate_ex(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_authenticate_ex.

bt_hci_cancel_find_devices Function

File

hci.h

C

```
bt_bool bt_hci_cancel_find_devices(bt_hci_cmd_callback_fp cb);
```

Description

brief Stop inquiry

bt_hci_cancel_request_remote_name Function

File

hci.h

C

```
bt_bool bt_hci_cancel_request_remote_name(bt_bdaddr_p pbdaddr);
```

Description

brief Cancel remote device name request

bt_hci_cancel_send_acl_data Function

File

hci.h

C

```
void bt_hci_cancel_send_acl_data(bt_byte* data);
```

Description

brief Cancel sending data over ACL connection

bt_hci_connect Function

File

hci.h

C

```
bt_bool bt_hci_connect(bt_bdaddr_p dest, bt_uint packet_type, bt_byte pg_scan_rpt_mode, bt_byte
role_switch, bt_uint acl_config, bt_hci_connect_callback_fp callback, void* param);
```

Description

brief Connect to a remote device. ingroup hci

details This function tries to establish an HCI connection with a remote device specified by the Bluetooth address c dest. Upon completion, the callback function specified by the c callback parameter is called.

param dest Bluetooth address of the remote device. param packet_type param role_switch param acl_config param callback Pointer to a callback function that is called when the connect operation completes. param param Pointer to arbitrary data that is to be passed to the callback function.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_connect_sco Function

File

hci.h

C

```
bt_bool bt_hci_connect_sco(bt_hci_conn_state_t* pconn, bt_ulong tx_bandwidth, bt_ulong rcv_bandwidth,
bt_uint max_latency, bt_uint content_format, bt_byte retransmission_effort, bt_uint packet_type,
bt_hci_connect_callback_fp cb, void * param);
```

Description

This is function bt_hci_connect_sco.

bt_hci_ctrl_register_data_listener Function

File

hci_ctrl_state.h

C

```
bt_bool bt_hci_ctrl_register_data_listener(bt_hci_ctrl_listener_t* listener, bt_byte listener_type);
```

Description

This is function bt_hci_ctrl_register_data_listener.

bt_hci_ctrl_register_listener Function

File

hci_ctrl_state.h

C

```
bt_bool bt_hci_ctrl_register_listener(bt_hci_ctrl_listener_t* listener);
```

Description

This is function bt_hci_ctrl_register_listener.

bt_hci_ctrl_unregister_listener Function

File

hci_ctrl_state.h

C

```
void bt_hci_ctrl_unregister_listener(bt_hci_ctrl_listener_t* listener);
```

Description

This is function bt_hci_ctrl_unregister_listener.

bt_hci_disconnect Function

File

hci.h

C

```
bt_bool bt_hci_disconnect(bt_hci_conn_state_t * pconn);
```

Description

brief Abort connection ingroup hci

bt_hci_evt_authentication_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_authentication_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_authentication_complete_handler.

bt_hci_evt_change_conn_link_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_change_conn_link_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_change_conn_link_complete_handler.

bt_hci_evt_command_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_command_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_command_complete_handler.

bt_hci_evt_command_status_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_command_status_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_command_status_handler.

bt_hci_evt_conn_packet_type_changed_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_conn_packet_type_changed_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_conn_packet_type_changed_handler.

bt_hci_evt_connection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_connection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_connection_complete_handler.

bt_hci_evt_connection_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_connection_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_connection_request_handler.

bt_hci_evt_data_buffer_overflow_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_data_buffer_overflow_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_data_buffer_overflow_handler.

bt_hci_evt_default_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_default_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_default_handler.

bt_hci_evt_disconnection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_disconnection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_disconnection_complete_handler.

bt_hci_evt_encryption_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_encryption_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_encryption_change_handler.

bt_hci_evt_extended_inquiry_result_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_extended_inquiry_result_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_extended_inquiry_result_handler.

bt_hci_evt_flow_specification_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_flow_specification_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_flow_specification_complete_handler.

bt_hci_evt_flush_occured_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_flush_occured_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_flush_occured_handler.

bt_hci_evt_hardware_error_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_hardware_error_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_hardware_error_handler.

bt_hci_evt_inquiry_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_complete_handler.

bt_hci_evt_inquiry_result_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_result_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_result_handler.

bt_hci_evt_inquiry_result_with_rssi_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_result_with_rssi_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_result_with_rssi_handler.

bt_hci_evt_link_key_notification_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_link_key_notification_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_link_key_notification_handler.

bt_hci_evt_link_key_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_link_key_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_link_key_request_handler.

bt_hci_evt_loopback_command_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_loopback_command_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_loopback_command_handler.

bt_hci_evt_master_link_key_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_master_link_key_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_master_link_key_complete_handler.

bt_hci_evt_max_slots_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_max_slots_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_max_slots_change_handler.

bt_hci_evt_mode_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_mode_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_mode_change_handler.

bt_hci_evt_num_of_completed_packets_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_num_of_completed_packets_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_num_of_completed_packets_handler.

bt_hci_evt_page_scan_repet_mode_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_page_scan_repet_mode_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_page_scan_repet_mode_change_handler.

bt_hci_evt_pin_code_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_pin_code_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_pin_code_request_handler.

bt_hci_evt_qos_setup_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_qos_setup_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_qos_setup_complete_handler.

bt_hci_evt_qos_violation_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_qos_violation_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_qos_violation_handler.

bt_hci_evt_read_clock_offset_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_clock_offset_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_clock_offset_complete_handler.

bt_hci_evt_read_rmt_ext_features_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_ext_features_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_ext_features_comp_handler.

bt_hci_evt_read_rmt_sup_features_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_sup_features_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_sup_features_comp_handler.

bt_hci_evt_read_rmt_version_info_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_version_info_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_version_info_comp_handler.

bt_hci_evt_remote_name_request_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_remote_name_request_complete_handler(bt_hci_event_p pevt, bt_uint params_len);
```

Description

This is function bt_hci_evt_remote_name_request_complete_handler.

bt_hci_evt_return_link_keys_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_return_link_keys_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_return_link_keys_handler.

bt_hci_evt_role_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_role_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_role_change_handler.

bt_hci_evt_synch_connection_changed_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_synch_connection_changed_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_synch_connection_changed_handler.

bt_hci_evt_synch_connection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_synch_connection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_synch_connection_complete_handler.

bt_hci_exit_park_state Function

File

hci.h

C

```
bt_bool bt_hci_exit_park_state(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb);
```

Description

brief Exit park state

bt_hci_exit_sniff_mode_ex Function

File

hci.h

C

```
bt_bool bt_hci_exit_sniff_mode_ex(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_exit_sniff_mode_ex.

bt_hci_find_devices Function

File

hci.h

C

```
bt_bool bt_hci_find_devices(bt_byte max_responses, bt_byte length, bt_hci_inquiry_callback_fp cb);
```

Description

brief Start inquiry

bt_hci_find_devices_ex Function

File

hci.h

C

```
bt_bool bt_hci_find_devices_ex(bt_byte max_responses, bt_byte length, bt_byte mode,
bt_hci_inquiry_callback_fp cb);
```

Description

This is function bt_hci_find_devices_ex.

bt_hci_free_command Function

File

hci.h

C

```
void bt_hci_free_command(bt_hci_command_p cmd);
```

Description

brief Free HCI command. ingroup hci

bt_hci_free_data_buffer Function

File

hci_data_buffer.h

C

```
void bt_hci_free_data_buffer(bt_hci_data_p p);
```

Description

This is function bt_hci_free_data_buffer.

bt_hci_get_evt_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_bdaddr(bt_hci_event_p pevt, bt_bdaddr_p pvalue, bt_int_p poffset);
```

Description

brief Get bd address parameter from HCI event ingroup hci

bt_hci_get_evt_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_byte(bt_hci_event_p pevt, bt_byte* pvalue, bt_int* poffset);
```

Description

brief Get byte parameter from HCI event ingroup hci

bt_hci_get_evt_param_devclass Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_devclass(bt_hci_event_p pevt, bt_long_p pvalue, bt_int_p poffset);
```

Description

brief Get class of device parameter from HCI event ingroup hci

bt_hci_get_evt_param_int Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_int(bt_hci_event_p pevt, bt_int* pvalue, bt_int* poffset);
```

Description

brief Get int parameter from HCI event ingroup hci

bt_hci_get_evt_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_linkkey(bt_hci_event_p pevt, bt_linkkey_p pvalue, bt_int_p poffset);
```

Description

brief Get link key parameter from HCI event ingroup hci

bt_hci_get_evt_param_long Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_long(bt_hci_event_p pevt, bt_long* pvalue, bt_int* poffset);
```

Description

brief Get long parameter from HCI event ingroup hci

bt_hci_get_evt_param_uint Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_uint(bt_hci_event_p pevt, bt_uint* pvalue, bt_int* poffset);
```

Description

brief Get unsigned int parameter from HCI event ingroup hci

bt_hci_get_evt_param_ulong Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_ulong(bt_hci_event_p pevt, bt_ulong* pvalue, bt_int* poffset);
```

Description

brief Get unsigned long parameter from HCI event ingroup hci

bt_hci_get_last_cmd_status Function

File

hci.h

C

```
bt_byte bt_hci_get_last_cmd_status();
```

Description

This is function bt_hci_get_last_cmd_status.

bt_hci_get_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_get_param_bdaddr(bt_hci_command_p pcm, bt_bdaddr_p pvalue, bt_int_p poffset);
```

Description

brief Get BD address parameter from HCI command ingroup hci

bt_hci_get_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_get_param_byte(bt_hci_command_p pcmd, bt_byte* pvalue, bt_int* offset);
```

Description

brief Get byte parameter from HCI command ingroup hci

bt_hci_get_param_int Function

File

hci.h

C

```
bt_bool bt_hci_get_param_int(bt_hci_command_p pcmd, bt_int* pvalue, bt_int* poffset);
```

Description

brief Get int parameter from HCI command ingroup hci

bt_hci_get_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_get_param_linkkey(bt_hci_command_p pcmd, bt_byte_p pvalue, bt_int_p poffset);
```

Description

brief Get link key parameter from HCI command ingroup hci

bt_hci_get_param_long Function

File

hci.h

C

```
bt_bool bt_hci_get_param_long(bt_hci_command_p pcmd, bt_long* pvalue, bt_int* poffset);
```

Description

brief Get long parameter from HCI command ingroup hci

bt_hci_get_recv_buffer Function

File

hci.h

C

```
bt_byte* bt_hci_get_recv_buffer();
```

Description

This is function bt_hci_get_recv_buffer.

bt_hci_get_recv_buffer_len Function

File

hci.h

C

```
bt_int bt_hci_get_recv_buffer_len();
```

Description

This is function bt_hci_get_recv_buffer_len.

bt_hci_get_send_buffer Function

File

hci.h

C

```
bt_byte* bt_hci_get_send_buffer();
```

Description

This is function bt_hci_get_send_buffer.

bt_hci_get_send_buffer_len Function

File

hci.h

C

```
bt_int bt_hci_get_send_buffer_len();
```

Description

This is function bt_hci_get_send_buffer_len.

bt_hci_init Function

File

hci.h

C

```
void bt_hci_init();
```

Description

brief Initialize the HCI layer. ingroup hci

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init.

This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

bt_hci_init_data_buffers Function

File

hci_data_buffer.h

C

```
void bt_hci_init_data_buffers();
```

Description

This is function bt_hci_init_data_buffers.

bt_hci_init_data_queues Function

File

hci_data_queue.h

C

```
void bt_hci_init_data_queues();
```

Description

This is function bt_hci_init_data_queues.

bt_hci_init_linkkey_buffers Function

File

hci_linkkey_buffer.h

C

```
void bt_hci_init_linkkey_buffers();
```

Description

This is function bt_hci_init_linkkey_buffers.

bt_hci_le_add_device_to_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_add_device_to_white_list(bt_byte address_type, bt_bdaddr_t* address,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_add_device_to_white_list.

bt_hci_le_advertising_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_add(bt_byte data_type, const bt_byte* data, bt_byte length, bt_hci_command_p
pcmd);
```

Description

This is function bt_hci_le_advertising_add.

bt_hci_le_advertising_add_local_name Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_add_local_name(const char* local_name, bt_hci_command_t* pcmd);
```

Description

This is function bt_hci_le_advertising_add_local_name.

bt_hci_le_advertising_device_id_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_device_id_add(bt_uint vendor_id_source, bt_uint vendor_id, bt_uint
product_id, bt_uint version, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_device_id_add.

bt_hci_le_advertising_flags_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_flags_add(bt_byte flags, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_flags_add.

bt_hci_le_advertising_get_local_name Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_get_local_name(const bt_byte* data, bt_byte data_len, const bt_byte**
local_name, bt_byte* name_len, bt_byte* name_type);
```

Description

This is function bt_hci_le_advertising_get_local_name.

bt_hci_le_advertising_tx_power_level_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_tx_power_level_add(bt_byte tx_power_level, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_tx_power_level_add.

bt_hci_le_advertising_uuid128_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid128_add(bt_byte data_type, const bt_uuid_t* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid128_add.

bt_hci_le_advertising_uuid16_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid16_add(bt_byte data_type, const bt_uint* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid16_add.

bt_hci_le_advertising_uuid32_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid32_add(bt_byte data_type, const bt_uuid32* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid32_add.

bt_hci_le_advertising_vendor_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_vendor_add(bt_uint vendor_id, bt_byte* data, bt_byte data_len,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_vendor_add.

bt_hci_le_allocate_set_advertising_data_command Function

File

hci_le.h

C

```
bt_hci_command_t* bt_hci_le_allocate_set_advertising_data_command(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_allocate_set_advertising_data_command.

bt_hci_le_cancel_find_devices Function

File

hci_le.h

C

```
bt_bool bt_hci_le_cancel_find_devices();
```

Description

This is function bt_hci_le_cancel_find_devices.

bt_hci_le_clear_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_clear_white_list(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_clear_white_list.

bt_hci_le_connect_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_connect_ex(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte
peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void * param);
```

Description

This is function bt_hci_le_connect_ex.

bt_hci_le_enable Function

File

hci_le.h

C

```
bt_bool bt_hci_le_enable(bt_bool enabled, bt_bool simultaneous_enabled, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_enable.

bt_hci_le_encrypt Function

File

hci_le.h

C

```
bt_bool bt_hci_le_encrypt(const bt_byte* key, bt_byte key_len, const bt_byte* data, bt_byte data_len,
bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_encrypt.

bt_hci_le_find_devices Function

File

hci_le.h

C

```
bt_bool bt_hci_le_find_devices(bt_byte filter_duplicates, bt_hci_le_scan_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_find_devices.

bt_hci_le_get_connect_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_get_connect_parameters(bt_hci_le_connect_parameters_t* params);
```

Description

This is function bt_hci_le_get_connect_parameters.

bt_hci_le_init Function

File

hci_private.h

C

```
void bt_hci_le_init(bt_hci_le_ctrl_state_t* le_ctrl_state);
```

Description

This is function bt_hci_le_init.

bt_hci_le_ltk_negative_reply Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ltk_negative_reply(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_ltk_negative_reply.

bt_hci_le_ltk_reply Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ltk_reply(struct _bt_hci_conn_state_s* pconn, const bt_byte* long_term_key,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_ltk_reply.

bt_hci_le_rand Function

File

hci_le.h

C

```
bt_bool bt_hci_le_rand(bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_rand.

bt_hci_le_read_channel_map Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_channel_map(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_channel_map.

bt_hci_le_read_remote_used_features Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_remote_used_features(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_remote_used_features.

bt_hci_le_read_support Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_support(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_support.

bt_hci_le_read_white_list_size Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_white_list_size(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_white_list_size.

bt_hci_le_receiver_test Function

File

hci_le.h

C

```
bt_bool bt_hci_le_receiver_test(bt_byte frequency, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_receiver_test.

bt_hci_le_remove_device_from_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_remove_device_from_white_list(bt_byte address_type, bt_bdaddr_t* address,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_remove_device_from_white_list.

bt_hci_le_set_adevrtising_enable_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_adevrtising_enable_ex(bt_byte adv_enable, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_le_set_adevrtising_enable_ex.

bt_hci_le_set_advertising_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_advertising_parameters(bt_uint adv_interval_min, bt_uint adv_interval_max, bt_byte
adv_type, bt_byte own_address_type, bt_byte direct_address_type, bt_bdaddr_t* direct_address, bt_byte
adv_channel_map, bt_byte adv_filter_policy, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_advertising_parameters.

bt_hci_le_set_connect_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_connect_parameters(bt_hci_le_connect_parameters_t* params);
```

Description

This is function bt_hci_le_set_connect_parameters.

bt_hci_le_set_host_channel_classification Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_host_channel_classification(const bt_byte* channel_map, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_host_channel_classification.

bt_hci_le_set_random_address Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_random_address(bt_bdaddr_t* bdaddr, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_random_address.

bt_hci_le_set_scan_enable Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_scan_enable(bt_byte scan_enable, bt_byte filter_duplicates, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_hci_le_set_scan_enable.

bt_hci_le_set_scan_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_scan_parameters(bt_byte scan_type, bt_uint scan_interval, bt_uint scan_window,
bt_byte own_address_type, bt_byte filter_policy, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_scan_parameters.

bt_hci_le_start_encryption Function

File

hci_le.h

C

```
bt_bool bt_hci_le_start_encryption(struct _bt_hci_conn_state_s* pconn, const bt_byte* random_number,
bt_uint encrypted_diversifier, const bt_byte* long_term_key, bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_start_encryption.

bt_hci_le_supported Function

File

hci_le.h

C

```
bt_bool bt_hci_le_supported();
```

Description

This is function bt_hci_le_supported.

bt_hci_le_test_end Function

File

hci_le.h

C

```
bt_bool bt_hci_le_test_end(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_test_end.

bt_hci_le_transmitter_test Function

File

hci_le.h

C

```
bt_bool bt_hci_le_transmitter_test(bt_byte frequency, bt_byte test_data_len, bt_byte packet_payload,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_transmitter_test.

bt_hci_le_update_connection Function

File

hci_le.h

C

```
bt_bool bt_hci_le_update_connection(struct _bt_hci_conn_state_s* pconn, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_hci_cmd_callback_fp cb, void * param);
```

Description

This is function bt_hci_le_update_connection.

bt_hci_le_write_support Function

File

hci_le.h

C

```
bt_bool bt_hci_le_write_support(bt_bool enabled, bt_bool simultaneous_enabled, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_write_support.

bt_hci_listen Function

File

hci.h

C

```
bt_bool bt_hci_listen(bt_hci_connect_callback_fp cb, void* param);
```

Description

brief Listen for incoming connections. ingroup hci

details

param callback Pointer to a callback function that is called when a new incoming connection has been established. param param Pointer to arbitrary data that is to be passed to the callback function.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_listen_sco Function

File

hci.h

C

```
bt_bool bt_hci_listen_sco(bt_ulong tx_bandwidth, bt_ulong rcv_bandwidth, bt_uint max_latency, bt_uint
content_format, bt_byte retransmission_effort, bt_uint packet_type, bt_hci_connect_callback_fp cb, void *
param);
```

Description

This is function bt_hci_listen_sco.

bt_hci_param_eir_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_add(bt_byte data_type, const bt_byte* data, bt_byte length, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_add.

bt_hci_param_eir_device_id_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_device_id_add(bt_uint vendor_id_source, bt_uint vendor_id, bt_uint product_id,
bt_uint version, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_device_id_add.

bt_hci_param_eir_local_name_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_local_name_add(const char* local_name, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_local_name_add.

bt_hci_param_eir_uuid128_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid128_add(bt_byte data_type, const bt_uuid_t* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid128_add.

bt_hci_param_eir_uuid16_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid16_add(bt_byte data_type, const bt_uint* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid16_add.

bt_hci_param_eir_uuid32_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid32_add(bt_byte data_type, const bt_uuid32* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid32_add.

bt_hci_param_eir_vendor_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_vendor_add(bt_uint vendor_id, bt_byte* data, bt_byte data_len, bt_hci_command_p
pcmd);
```

Description

This is function bt_hci_param_eir_vendor_add.

bt_hci_param_tx_power_level_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_tx_power_level_add(bt_byte tx_power_level, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_tx_power_level_add.

bt_hci_park_state Function

File

hci.h

C

```
bt_bool bt_hci_park_state(bt_hci_conn_state_t * pconn, bt_int min_interval, bt_int max_interval,
bt_hci_cmd_callback_fp cb);
```

Description

brief Put local device to park state

bt_hci_read_inquiry_mode Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_mode(bt_hci_read_inquiry_mode_callback_fp cb);
```

Description

brief Get current inquiry mode

bt_hci_read_inquiry_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_scan_activity(bt_hci_read_inquiry_scan_activity_callback_fp cb);
```

Description

brief Get current inquiry scan activity configuration

bt_hci_read_inquiry_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_scan_type(bt_hci_read_inquiry_scan_type_callback_fp cb);
```

Description

brief Get current inquiry scan type

bt_hci_read_page_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_activity(bt_hci_read_page_scan_activity_callback_fp cb);
```

Description

brief Get current page scan activity configuration

bt_hci_read_page_scan_period_mode Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_period_mode(bt_hci_read_page_scan_period_mode_callback_fp cb);
```

Description

brief Get current page scan period mode

bt_hci_read_page_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_type(bt_hci_read_page_scan_type_callback_fp cb);
```

Description

brief Get current page scan type

bt_hci_read_page_timeout Function

File

hci.h

C

```
bt_bool bt_hci_read_page_timeout(bt_hci_read_page_timeout_callback_fp cb);
```

Description

brief Get current page timeout

bt_hci_register_listener Function

File

hci_conn_state.h

C

```
bt_bool bt_hci_register_listener(bt_hci_conn_state_t* pconn, bt_hci_listener_t* listener);
```

Description

This is function bt_hci_register_listener.

bt_hci_reject_pin_code Function

File

hci.h

C

```
void bt_hci_reject_pin_code(bt_bdaddr_p pbdaddr_remote);
```

Description

This is function bt_hci_reject_pin_code.

bt_hci_request_remote_name Function

File

hci.h

C

```
bt_bool bt_hci_request_remote_name(bt_bdaddr_p pbdaddr, bt_byte pg_scan_rpt_mode, bt_int clock_offset,
bt_hci_request_remote_name_callback_fp cb);
```

Description

brief Request remote device's name

bt_hci_role_change_ex Function

File

hci.h

C

```
bt_bool bt_hci_role_change_ex(bt_bdaddr_p pbdaddr, bt_byte role, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Change local device's role

bt_hci_send_acl_data Function

File

hci.h

C

```
bt_bool bt_hci_send_acl_data(bt_hci_hconn_t h, bt_byte_p pdata, bt_int len, bt_hci_data_callback_fp cb,
void * param);
```

Description

brief Send data over ACL connection

bt_hci_send_cmd Function

File

hci.h

C

```
bt_bool bt_hci_send_cmd(bt_hci_command_p pcmd);
```

Description

brief Send a command to local device

bt_hci_send_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_send_linkkey(const bt_bdaddr_t* bdaddr_remote, const bt_linkkey_t* link_key,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_send_linkkey.

bt_hci_send_sco_data Function

File

hci.h

C

```
bt_bool bt_hci_send_sco_data(bt_hci_hconn_t h, bt_byte_p pdata, bt_uint len, bt_hci_data_callback_fp cb,
void * param);
```

Description

brief Send data over SCO connection

bt_hci_set_encryption_ex Function

File

hci.h

C

```
bt_bool bt_hci_set_encryption_ex(bt_hci_conn_state_t* pconn, bt_byte encryption_enable,
bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Set connection encryption

bt_hci_set_event_listener Function

File

hci_ctrl_state.h

C

```
void bt_hci_set_event_listener(bt_hci_event_listener_fp callback);
```

Description

listener is triggered by the following events: HCI_EVT_AUTHENTICATION_COMPLETE HCI_EVT_CONNECTION_COMPLETE HCI_EVT_DISCONNECTION_COMPLETE HCI_EVT_ROLE_CHANGE HCI_EVT_MODE_CHANGE

bt_hci_set_incoming_connection_role Function

File

hci.h

C

```
void bt_hci_set_incoming_connection_role(bt_byte role);
```

Description

This is function bt_hci_set_incoming_connection_role.

bt_hci_set_scan Function

File

hci.h

C

```
bt_bool bt_hci_set_scan(bt_bool discoverable, bt_ulong discoverable_period, bt_bool connectable, bt_ulong
connectable_period, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_set_scan.

bt_hci_sniff_mode_ex Function

File

hci.h

C

```
bt_bool bt_hci_sniff_mode_ex(bt_hci_conn_state_t * pconn, bt_int min_interval, bt_int max_interval, bt_int
attempt_slots, bt_int timeout, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_sniff_mode_ex.

bt_hci_sniff_subrating_ex Function

File

hci.h

C

```
bt_bool bt_hci_sniff_subrating_ex(bt_hci_conn_state_t* pconn, bt_uint max_latency, bt_uint
min_remote_timeout, bt_uint min_local_timeout, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_sniff_subrating_ex.

bt_hci_start Function

File

hci.h

C

```
bt_bool bt_hci_start(bt_hci_start_callback_fp callback, void* callback_param, bt_byte enable_scan);
```

Description

brief Start HCI layer. ingroup hci

details This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps:

1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv.
2. Reset and configure the controller.

Upon completion of controller initialization the callback function passed in the c callback parameter is called.

param callback Completion callback. Called when controller initialization is complete. param callback_param A pointer to arbitrary data to be passed to the c callback callback. param enable_scan This is a bitmask that defines which scans are enabled during the controller configuration. This value can be a combination of the following values: li HCI_SCAN_INQUIRY (the controller is discoverable) li HCI_SCAN_PAGE (the controller is connectable)

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_start_no_init Function

File

hci.h

C

```
void bt_hci_start_no_init();
```

Description

brief Start HCI layer without controller configuration. ingroup hci

details This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would call bt_hci_init(), bt_hci_start_no_init() and then load configuration values. Once the vendor specific configuration is done, the application will re-initialize the HCI transport and perform full start of the stack with bt_sys_init() and bt_sys_start().

bt_hci_stop Function

File

hci.h

C

```
void bt_hci_stop(bt_hci_stop_callback_fp callback, void* callback_param);
```

Description

brief Stop HCI layer. ingroup hci

details This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack.

param callback Completion callback. Called when the HCI layer has been stopped. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_hci_transport_recv_packet Function

File

hci_transport.h

C

```
void bt_hci_transport_recv_packet(bt_byte* buffer, bt_uint len, bt_hci_transport_recv_packet_callback_fp
callback);
```

Description

This is function bt_hci_transport_recv_packet.

bt_hci_transport_send_cmd Function

File

hci_transport.h

C

```
void bt_hci_transport_send_cmd(const bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
```

Description

deprecated

bt_hci_transport_send_data Function

File

hci_transport.h

C

```
void bt_hci_transport_send_data(bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
```

Description

This is function bt_hci_transport_send_data.

bt_hci_transport_send_packet Function

File

hci_transport.h

C

```
void bt_hci_transport_send_packet(const bt_byte* buffer, bt_uint len,
bt_hci_transport_send_packet_callback_fp callback);
```

Description

This is function bt_hci_transport_send_packet.

bt_hci_transport_set_transport Function

File

hci_transport.h

C

```
void bt_hci_transport_set_transport(const hci_transport_t* transport);
```

Description

This is function bt_hci_transport_set_transport.

bt_hci_unregister_listener Function

File

hci_conn_state.h

C

```
void bt_hci_unregister_listener(bt_hci_conn_state_t* pconn, bt_hci_listener_t* listener);
```

Description

This is function bt_hci_unregister_listener.

bt_hci_write_default_link_policy_settings Function

File

hci.h

C

```
bt_bool bt_hci_write_default_link_policy_settings(bt_uint settings, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_write_default_link_policy_settings.

bt_hci_write_eir Function

File

hci_eir.h

C

```
bt_bool bt_hci_write_eir(bt_hci_command_p pcmd, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_write_eir.

bt_hci_write_inquiry_mode Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_mode(bt_byte scanMode, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry mode

bt_hci_write_inquiry_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_scan_activity(bt_uint interval, bt_uint window, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry scan activity

bt_hci_write_inquiry_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_scan_type(bt_byte scanType, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry scan type

bt_hci_write_local_name Function

File

hci.h

C

```
bt_bool bt_hci_write_local_name(const char * device_name, bt_hci_cmd_callback_fp cb);
```

Description

brief Write local device name ingroup hci

details The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.

bt_hci_write_page_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_activity(bt_uint interval, bt_uint window, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan activity

bt_hci_write_page_scan_period_mode Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_period_mode(bt_byte mode, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan period mode

bt_hci_write_page_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_type(bt_byte scanType, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan type

bt_hci_write_page_timeout Function

File

hci.h

C

```
bt_bool bt_hci_write_page_timeout(bt_uint timeout, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page timeout

bt_hcitr_3wire_start Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_start();
```

Description

This is function bt_hcitr_3wire_start.

bt_hcitr_bcsp_init_ex Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_init_ex(bt_ulong ack_timeout);
```

Description

This is function bt_hcitr_bcsp_init_ex.

bt_hcitr_bcsp_reset_ex Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_reset_ex(bt_ulong ack_timeout);
```

Description

This is function bt_hcitr_bcsp_reset_ex.

bt_hcitr_bcsp_start Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_start();
```

Description

This is function bt_hcitr_bcsp_start.

bt_hcitr_packet_init Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_init();
```

Description

This is function bt_hcitr_packet_init.

bt_hcitr_packet_reset Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_reset();
```

Description

This is function bt_hcitr_packet_reset.

bt_hcitr_packet_start Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_start();
```

Description

This is function bt_hcitr_packet_start.

bt_hcitr_tih4_init Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_init(bt_hcitr_tih4_power_callback_fp callback);
```

Description

This is function bt_hcitr_tih4_init.

bt_hcitr_tih4_reset Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_reset();
```

Description

This is function bt_hcitr_tih4_reset.

bt_hcitr_tih4_start Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_start();
```

Description

This is function bt_hcitr_tih4_start.

bt_hcitr_tih4_wake_up Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_wake_up();
```

Description

This is function bt_hcitr_tih4_wake_up.

bt_hcitr_uart_init Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_init();
```

Description

brief Initialize HCI UART (H4) transport protocol ingroup hcitr_uart

details This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.

bt_hcitr_uart_reset Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_reset();
```

Description

brief Re-initialize HCI UART (H4) transport protocol ingroup hcitr_uart

details This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.

bt_hcitr_uart_start Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_start();
```

Description

brief Start HCI UART (H4) transport protocol ingroup hcitr_uart

details This function starts the transport, i.e., makes it able to receive and send packets.

hci_allocate_conn_state Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_allocate_conn_state(struct _bt_hci_ctrl_state_s * pctrl);
```

Description

This is function hci_allocate_conn_state.

hci_check_aux_info Function

File

hci_private.h

C

```
bt_bool hci_check_aux_info(bt_long bdaddr_l, bt_long bdaddr_m);
```

Description

From hci_aux_info.c

hci_cq_find_by_bdaddr_and_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_bdaddr_and_opcode(bt_queue_element_t* head, bt_bdaddr_t bdaddrFind, bt_int opcode);
```

Description

This is function hci_cq_find_by_bdaddr_and_opcode.

hci_cq_find_by_hconn Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_hconn(bt_queue_element_t* head, bt_hci_hconn_t hconn);
```

Description

This is function hci_cq_find_by_hconn.

hci_cq_find_by_hconn_and_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_hconn_and_opcode(bt_queue_element_t* head, bt_hci_hconn_t hconnFind, bt_int opcode);
```

Description

This is function hci_cq_find_by_hconn_and_opcode.

hci_cq_find_by_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_opcode(bt_queue_element_t* head, bt_int opcode);
```

Description

This is function hci_cq_find_by_opcode.

hci_get_conn_state Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_get_conn_state(struct _bt_hci_ctrl_state_s * pctrl, bt_hci_hconn_t h);
```

Description

This is function hci_get_conn_state.

hci_sleep Function

File

hci.h

C

```
void hci_sleep(pf_hci_sleep_callback cb);
```

Description

brief Put local device to sleep

hci_wakeup Function

File

hci.h

C

```
void hci_wakeup(pf_hci_wakeup_callback cb);
```

Description

brief Awaken local device

is_bdaddr_command Function

File

hci_utils.h

C

```
bt_bool is_bdaddr_command(bt_int opcode);
```

Description

This is function is_bdaddr_command.

is_hconn_command Function

File

hci_utils.h

C

```
bt_bool is_hconn_command(bt_int opcode);
```

Description

This is function is_hconn_command.

_bt_hci_ctrl_notify_data_listeners Function

File

hci_private.h

C

```
void _bt_hci_ctrl_notify_data_listeners(bt_hci_conn_state_t* connection, const bt_byte* data, bt_int len,
bt_bool sent);
```

Description

This is function _bt_hci_ctrl_notify_data_listeners.

_bt_hci_ctrl_notify_listeners Function

File

hci_private.h

C

```
void _bt_hci_ctrl_notify_listeners(bt_uint event_id, void* params);
```

Description

This is function _bt_hci_ctrl_notify_listeners.

_bt_hci_get_tick_count Function

File

hci_private.h

C

```
bt_ulong _bt_hci_get_tick_count();
```

Description

This is function _bt_hci_get_tick_count.

_bt_hci_init_signal Function

File

hci_signal.h

C

```
void _bt_hci_init_signal();
```

Description

This is function _bt_hci_init_signal.

_bt_hci_init_timer Function

File

hci_private.h

C

```
void _bt_hci_init_timer();
```

Description

From hci_timer.c

_bt_hci_init_transport Function

File

hci_private.h

C

```
void _bt_hci_init_transport();
```

Description

This is function _bt_hci_init_transport.

_bt_hci_le_command_complete_handler Function

File

hci_private.h

C

```
void _bt_hci_le_command_complete_handler(bt_int status, bt_hci_command_t* pcmd, bt_hci_event_t* pevt);
```

Description

This is function _bt_hci_le_command_complete_handler.

_bt_hci_notify_listeners Function

File

hci_private.h

C

```
void _bt_hci_notify_listeners(bt_hci_conn_state_t* pconn, bt_byte event_id, bt_hci_event_e* params);
```

Description

This is function _bt_hci_notify_listeners.

_bt_hci_set_signal Function

File

hci_signal.h

C

```
void _bt_hci_set_signal();
```

Description

This is function _bt_hci_set_signal.

_hci_allocate_buffers Function

File

hci_private.h

C

```
void _hci_allocate_buffers();
```

Description

Defined by OEM through library configuration

_hci_allocate_cmd Function

File

hci_cmd_buffer.h

C

```
bt_hci_command_t* _hci_allocate_cmd();
```

Description

This is function _hci_allocate_cmd.

_hci_free_cmd Function

File

hci_cmd_buffer.h

C

```
void _hci_free_cmd(bt_hci_command_t* pcmd);
```

Description

This is function _hci_free_cmd.

_hci_init_cmd_buffers Function

File

hci_cmd_buffer.h

C

```
void _hci_init_cmd_buffers();
```

Description

This is function _hci_init_cmd_buffers.

_hci_init_cmd_queues Function

File

hci_cmd_queue.h

C

```
void _hci_init_cmd_queues();
```

Description

This is function _hci_init_cmd_queues.

_hci_receive_start Function

File

hci_private.h

C

```
void _hci_receive_start();
```

Description

From hci_receive.c

_hci_recv_sco_data_packet Function

File

hci_private.h

C

```
void _hci_recv_sco_data_packet(bt_byte_p pbuf);
```

Description

From hci_sco.c

_hci_send_commands_from_queue Function

File

hci_private.h

C

```
void _hci_send_commands_from_queue();
```

Description

From hci_send.c

_hci_send_data Function

File

hci_private.h

C

```
bt_bool _hci_send_data(bt_hci_data_p pdata);
```

Description

This is function _hci_send_data.

_hci_send_data_fragment Function

File

hci_private.h

C

```
void _hci_send_data_fragment(bt_hci_data_p pdata);
```

Description

This is function _hci_send_data_fragment.

_hci_send_data_from_queue Function

File

hci_private.h

C

```
void _hci_send_data_from_queue();
```

Description

This is function _hci_send_data_from_queue.

_bt_le_evt_handler Function

File

hci_private.h

C

```
void _bt_le_evt_handler(bt_hci_event_t* evt);
```

Description

This is function _bt_le_evt_handler.

bt_hci_le_allocate_set_scan_response_data_command Function

File

hci_le.h

C

```
bt_hci_command_t* bt_hci_le_allocate_set_scan_response_data_command(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_allocate_set_scan_response_data_command.

bt_hci_le_cancel_connect_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_cancel_connect_ex(bt_hci_cmd_callback_fp cb, void * cb_param);
```

Description

This is function bt_hci_le_cancel_connect_ex.

bt_hci_le_ibeacon_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ibeacon_add(const bt_uuid_t* uuid, bt_uint major, bt_uint minor, bt_byte tx_power_level,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_ibeacon_add.

bt_hci_set_vendor_specific_event_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_set_vendor_specific_event_handler(bt_hci_event_handler_fp handler);
```

Description

This is function bt_hci_set_vendor_specific_event_handler.

bt_hcitr_3wire_init_ex Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_init_ex(bt_ulong ack_timeout, bt_bool data_integrity_check_enabled);
```

Description

This is function bt_hcitr_3wire_init_ex.

bt_hcitr_3wire_reset_ex Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_reset_ex(bt_ulong ack_timeout, bt_bool data_integrity_check_enabled);
```

Description

This is function bt_hcitr_3wire_reset_ex.

_bt_hci_set_init_flags Function

File

hci_private.h

C

```
void _bt_hci_set_init_flags(bt_byte flags);
```

Description

This is function _bt_hci_set_init_flags.

hci_get_conn_state_by_bdaddr Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_get_conn_state_by_bdaddr(struct _bt_hci_ctrl_state_s * pctrl, const bt_bdaddr_t*
pbdaddr_remote, bt_byte conn_type, bt_byte link_type);
```

Description

This is function hci_get_conn_state_by_bdaddr.

bt_hci_cancel_find_devices_ex Function

File

hci.h

C

```
bt_bool bt_hci_cancel_find_devices_ex(bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Stop inquiry

bt_hci_get_inquiry_response_tx_power_level Function

File

hci.h

C

```
bt_byte bt_hci_get_inquiry_response_tx_power_level();
```

Description

This is function bt_hci_get_inquiry_response_tx_power_level.

bt_hci_init_ex Function

File

hci.h

C

```
void bt_hci_init_ex(bt_byte default_link_policy);
```

Description

brief Initialize the HCI layer. ingroup hci

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex.

@param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE

To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

bt_hci_reset Function

File

hci.h

C

```
bt_bool bt_hci_reset(bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Reset controller. ingroup hci

details This function resets the BT controller.

param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_hci_send_pin_code_ex Function

File

hci.h

C

```
bt_bool bt_hci_send_pin_code_ex(bt_bdaddr_p pbdaddr_remote, const char* pin, bt_hci_cmd_callback_fp
callback, void* callback_param);
```

Description

This is function bt_hci_send_pin_code_ex.

bt_hci_set_scan_ex Function

File

hci.h

C

```
bt_bool bt_hci_set_scan_ex(bt_bool discoverable, bt_ulong discoverable_period, bt_byte discoverable_mode,
bt_bool connectable, bt_ulong connectable_period, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_set_scan_ex.

bt_hci_write_link_policy_settings Function

File

hci.h

C

```
bt_bool bt_hci_write_link_policy_settings(bt_hci_conn_state_t* conn, bt_uint settings,
bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_write_link_policy_settings.

bt_hci_write_local_name_ex Function

File

hci.h

C

```
bt_bool bt_hci_write_local_name_ex(const char * device_name, bt_hci_cmd_callback_fp cb, void*
callback_param);
```

Description

This is function bt_hci_write_local_name_ex.

bt_hcitr_3wire_cancel_recv_packet Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_cancel_recv_packet();
```

Description

This is function bt_hcitr_3wire_cancel_recv_packet.

HCI Data Types and Constants

bt_hci_add_param_hconn Macro

File

hci.h

C

```
#define bt_hci_add_param_hconn(pcmd, value) \
    bt_hci_add_param_int(pcmd, value)
```

Description

brief Add connection handle parameter to an HCI command ingroup hci

bt_hci_add_param_lap Macro

File

hci.h

C

```
#define bt_hci_add_param_lap(cmd, lap) bt_hci_add_param_cod(cmd, lap)
```

Description

This is macro bt_hci_add_param_lap.

bt_hci_add_param_uint Macro

File

hci.h

C

```
#define bt_hci_add_param_uint(cmd, value) bt_hci_add_param_int(cmd, (bt_int)(value))
```

Description

brief Add unsigned int parameter to an HCI command ingroup hci

bt_hci_add_param_ulong Macro

File

hci.h

C

```
#define bt_hci_add_param_ulong(cmd, value) bt_hci_add_param_long(cmd, (bt_long)(value))
```

Description

brief Add unsigned long parameter to an HCI command ingroup hci

bt_hci_authenticate Macro

File

hci.h

C

```
#define bt_hci_authenticate(pconn, cb) bt_hci_authenticate_ex(pconn, cb, NULL)
```

Description

This is macro bt_hci_authenticate.

bt_hci_exit_sniff_mode Macro

File

hci.h

C

```
#define bt_hci_exit_sniff_mode(pconn, cb) bt_hci_exit_sniff_mode_ex(pconn, cb, NULL)
```

Description

brief Cancel sniff mode

bt_hci_get_evt_param_hconn Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_hconn(pevt, pvalue, poffset) \
    bt_hci_get_evt_param_int(pevt, pvalue, poffset)
```

Description

brief Get connection handle parameter from HCI event ingroup hci

bt_hci_get_evt_param_uint Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_uint(pevt, pvalue, poffset) bt_hci_get_evt_param_int(pevt, (bt_int*)pvalue, poffset)
```

Description

This is macro bt_hci_get_evt_param_uint.

bt_hci_get_evt_param_ulong Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_ulong(pevt, pvalue, poffset) bt_hci_get_evt_param_long(pevt, (bt_long*)pvalue, poffset)
```

Description

This is macro bt_hci_get_evt_param_ulong.

bt_hci_get_param_hconn Macro

File

hci.h

C

```
#define bt_hci_get_param_hconn(pcmd, pvalue, poffset) bt_hci_get_param_int(pcmd, pvalue, poffset)
```

Description

brief Get connection handle parameter from HCI command ingroup hci

bt_hci_le_set_adevrtising_enable Macro

File

hci_le.h

C

```
#define bt_hci_le_set_adevrtising_enable(adv_enable, cb) bt_hci_le_set_adevrtising_enable_ex(adv_enable, cb, NULL)
```

Description

This is macro bt_hci_le_set_adevrtising_enable.

bt_hci_role_change Macro

File

hci.h

C

```
#define bt_hci_role_change(pbdaddr, role, cb) bt_hci_role_change_ex(pbdaddr, role, cb, NULL)
```

Description

This is macro bt_hci_role_change.

bt_hci_set_encryption Macro

File

hci.h

C

```
#define bt_hci_set_encryption(pconn, encryption_enable, cb) bt_hci_set_encryption_ex(pconn,
encryption_enable, cb, NULL)
```

Description

This is macro bt_hci_set_encryption.

bt_hci_sniff_mode Macro

File

hci.h

C

```
#define bt_hci_sniff_mode(pconn, min_interval, max_interval, attempt_slots, timeout, cb) \
    bt_hci_sniff_mode_ex(pconn, min_interval, max_interval, attempt_slots, timeout, cb, NULL)
```

Description

brief Put local device to sniff mode

bt_hci_sniff_subrating Macro

File

hci.h

C

```
#define bt_hci_sniff_subrating(pconn, max_latency, min_remote_timeout, min_local_timeout, cb) \
    bt_hci_sniff_subrating_ex(pconn, max_latency, min_remote_timeout, min_local_timeout, cb, NULL)
```

Description

This is macro bt_hci_sniff_subrating.

bt_hcitr_3wire_init Macro

File

hcitr_3wire.h

C

```
#define bt_hcitr_3wire_init bt_hcitr_3wire_init_ex(HCITR_3WIRE_DEFAULT_ACK_TIMEOUT, BT_FALSE)
```

Description

This is macro bt_hcitr_3wire_init.

bt_hcitr_3wire_reset Macro

File

hcitr_3wire.h

C

```
#define bt_hcitr_3wire_reset bt_hcitr_3wire_reset_ex(HCITR_3WIRE_DEFAULT_ACK_TIMEOUT, BT_FALSE)
```

Description

This is macro bt_hcitr_3wire_reset.

bt_hcitr_bcsp_init Macro

File

hcitr_bcsp.h

C

```
#define bt_hcitr_bcsp_init bt_hcitr_bcsp_init_ex(HCITR_BCSP_DEFAULT_ACK_TIMEOUT)
```

Description

This is macro bt_hcitr_bcsp_init.

bt_hcitr_bcsp_reset Macro

File

hcitr_bcsp.h

C

```
#define bt_hcitr_bcsp_reset bt_hcitr_bcsp_reset_ex(HCITR_BCSP_DEFAULT_ACK_TIMEOUT)
```

Description

This is macro bt_hcitr_bcsp_reset.

COD_MAJOR_AUDIO Macro

File

baseband.h

C

```
#define COD_MAJOR_AUDIO 0x0400
```

Description

This is macro COD_MAJOR_AUDIO.

COD_MAJOR_COMPUTER Macro

File

baseband.h

C

```
#define COD_MAJOR_COMPUTER 0x0100
```

Description

This is macro COD_MAJOR_COMPUTER.

COD_MAJOR_HEALTH Macro

File

baseband.h

C

```
#define COD_MAJOR_HEALTH 0x0900
```

Description

This is macro COD_MAJOR_HEALTH.

COD_MAJOR_IMAGING Macro

File

baseband.h

C

```
#define COD_MAJOR_IMAGING 0x0600
```

Description

This is macro COD_MAJOR_IMAGING.

COD_MAJOR_MISC Macro

File

baseband.h

C

```
#define COD_MAJOR_MISC 0x0000
```

Description

This is macro COD_MAJOR_MISC.

COD_MAJOR_NET_ACCESS_POINT Macro

File

baseband.h

C

```
#define COD_MAJOR_NET_ACCESS_POINT 0x0300
```

Description

This is macro COD_MAJOR_NET_ACCESS_POINT.

COD_MAJOR_PERIPHERAL Macro

File

baseband.h

C

```
#define COD_MAJOR_PERIPHERAL 0x0500
```

Description

This is macro COD_MAJOR_PERIPHERAL.

COD_MAJOR_PHONE Macro

File

baseband.h

C

```
#define COD_MAJOR_PHONE 0x0200
```

Description

This is macro COD_MAJOR_PHONE.

COD_MAJOR_TOY Macro

File

baseband.h

C

```
#define COD_MAJOR_TOY 0x0800
```

Description

This is macro COD_MAJOR_TOY.

COD_MAJOR_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MAJOR_UNCATEGORIZED 0x1f00
```

Description

This is macro COD_MAJOR_UNCATEGORIZED.

COD_MAJOR_WEARABLE Macro

File

baseband.h

C

```
#define COD_MAJOR_WEARABLE 0x0700
```

Description

This is macro COD_MAJOR_WEARABLE.

COD_MINOR_AV_CAMCORDER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_CAMCORDER (0x0d << 2)
```

Description

This is macro COD_MINOR_AV_CAMCORDER.

COD_MINOR_AV_CAR_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_CAR_AUDIO (0x08 << 2)
```

Description

This is macro COD_MINOR_AV_CAR_AUDIO.

COD_MINOR_AV_GAMING Macro

File

baseband.h

C

```
#define COD_MINOR_AV_GAMING (0x12 << 2)
```

Description

This is macro COD_MINOR_AV_GAMING.

COD_MINOR_AV_HANDSFREE Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HANDSFREE (0x02 << 2)
```

Description

This is macro COD_MINOR_AV_HANDSFREE.

COD_MINOR_AV_HEADPHONES Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HEADPHONES (0x06 << 2)
```

Description

This is macro COD_MINOR_AV_HEADPHONES.

COD_MINOR_AV_HEADSET Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HEADSET (0x01 << 2)
```

Description

This is macro COD_MINOR_AV_HEADSET.

COD_MINOR_AV_HIFI_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HIFI_AUDIO (0x0a << 2)
```

Description

This is macro COD_MINOR_AV_HIFI_AUDIO.

COD_MINOR_AV_LOUDSPEAKER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_LOUDSPEAKER (0x05 << 2)
```

Description

This is macro COD_MINOR_AV_LOUDSPEAKER.

COD_MINOR_AV_MICROPHONE Macro

File

baseband.h

C

```
#define COD_MINOR_AV_MICROPHONE (0x04 << 2)
```

Description

This is macro COD_MINOR_AV_MICROPHONE.

COD_MINOR_AV_PORTABLE_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_PORTABLE_AUDIO (0x07 << 2)
```

Description

This is macro COD_MINOR_AV_PORTABLE_AUDIO.

COD_MINOR_AV_RESERVED Macro

File

baseband.h

C

```
#define COD_MINOR_AV_RESERVED (0x03 << 2)
```

Description

This is macro COD_MINOR_AV_RESERVED.

COD_MINOR_AV_RESERVERD2 Macro

File

baseband.h

C

```
#define COD_MINOR_AV_RESERVERD2 (0x11 << 2)
```

Description

This is macro COD_MINOR_AV_RESERVERD2.

COD_MINOR_AV_SET_TOP_BOX Macro

File

baseband.h

C

```
#define COD_MINOR_AV_SET_TOP_BOX (0x09 << 2)
```

Description

This is macro COD_MINOR_AV_SET_TOP_BOX.

COD_MINOR_AV_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_AV_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_AV_UNCATEGORIZED.

COD_MINOR_AV_VCR Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VCR (0x0b << 2)
```

Description

This is macro COD_MINOR_AV_VCR.

COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER (0x0f << 2)
```

Description

This is macro COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER.

COD_MINOR_AV_VIDEO_CAMERA Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_CAMERA (0x0c << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_CAMERA.

COD_MINOR_AV_VIDEO_CONFERENCING Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_CONFERENCING (0x10 << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_CONFERENCING.

COD_MINOR_AV_VIDEO_MONITOR Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_MONITOR (0x0e << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_MONITOR.

COD_MINOR_COMPUTER_DESKTOP Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_DESKTOP (0x01 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_DESKTOP.

COD_MINOR_COMPUTER_HANDHELD Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_HANDHELD (0x04 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_HANDHELD.

COD_MINOR_COMPUTER_LAPTOP Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_LAPTOP (0x03 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_LAPTOP.

COD_MINOR_COMPUTER_PALMSIZED Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_PALMSIZED (0x05 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_PALMSIZED.

COD_MINOR_COMPUTER_SERVER Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_SERVER (0x02 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_SERVER.

COD_MINOR_COMPUTER_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_COMPUTER_UNCATEGORIZED.

COD_MINOR_COMPUTER_WEARABLE Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_WEARABLE (0x06 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_WEARABLE.

COD_MINOR_HEALTH_BPM Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_BPM (0x01 << 2)
```

Description

This is macro COD_MINOR_HEALTH_BPM.

COD_MINOR_HEALTH_DATA_DISPLAY Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_DATA_DISPLAY (0x07 << 2)
```

Description

This is macro COD_MINOR_HEALTH_DATA_DISPLAY.

COD_MINOR_HEALTH_GLUCOSE_METER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_GLUCOSE_METER (0x04 << 2)
```

Description

This is macro COD_MINOR_HEALTH_GLUCOSE_METER.

COD_MINOR_HEALTH_HEART_MONITOR Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_HEART_MONITOR (0x06 << 2)
```

Description

This is macro COD_MINOR_HEALTH_HEART_MONITOR.

COD_MINOR_HEALTH_PULSE_OXIMETER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_PULSE_OXIMETER (0x05 << 2)
```

Description

This is macro COD_MINOR_HEALTH_PULSE_OXIMETER.

COD_MINOR_HEALTH_THERMOMETER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_THERMOMETER (0x02 << 2)
```

Description

This is macro COD_MINOR_HEALTH_THERMOMETER.

COD_MINOR_HEALTH_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_HEALTH_UNCATEGORIZED.

COD_MINOR_HEALTH_WEIGHING_SCALE Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_WEIGHING_SCALE (0x03 << 2)
```

Description

This is macro COD_MINOR_HEALTH_WEIGHING_SCALE.

COD_MINOR_IMAGING_CAMERA Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_CAMERA (0x01 << 5)
```

Description

This is macro COD_MINOR_IMAGING_CAMERA.

COD_MINOR_IMAGING_DISPLAY Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_DISPLAY (0x01 << 4)
```

Description

This is macro COD_MINOR_IMAGING_DISPLAY.

COD_MINOR_IMAGING_PRINTER Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_PRINTER (0x01 << 7)
```

Description

This is macro COD_MINOR_IMAGING_PRINTER.

COD_MINOR_IMAGING_SCANNER Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_SCANNER (0x01 << 6)
```

Description

This is macro COD_MINOR_IMAGING_SCANNER.

COD_MINOR_IMAGING_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_IMAGING_UNCATEGORIZED.

COD_MINOR_LAN_01_17 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_01_17 (0x01 << 5)
```

Description

This is macro COD_MINOR_LAN_01_17.

COD_MINOR_LAN_17_33 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_17_33 (0x02 << 5)
```

Description

This is macro COD_MINOR_LAN_17_33.

COD_MINOR_LAN_33_50 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_33_50 (0x03 << 5)
```

Description

This is macro COD_MINOR_LAN_33_50.

COD_MINOR_LAN_50_67 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_50_67 (0x04 << 5)
```

Description

This is macro COD_MINOR_LAN_50_67.

COD_MINOR_LAN_67_83 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_67_83 (0x05 << 5)
```

Description

This is macro COD_MINOR_LAN_67_83.

COD_MINOR_LAN_83_99 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_83_99 (0x06 << 5)
```

Description

This is macro COD_MINOR_LAN_83_99.

COD_MINOR_LAN_FULLY_AVAILABLE Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_FULLY_AVAILABLE 0x00
```

Description

This is macro COD_MINOR_LAN_FULLY_AVAILABLE.

COD_MINOR_LAN_NO_SERVICE Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_NO_SERVICE (0x07 << 5)
```

Description

This is macro COD_MINOR_LAN_NO_SERVICE.

COD_MINOR_LAN_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_LAN_UNCATEGORIZED.

COD_MINOR_PERIPHERAL_CARD_READER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_CARD_READER (0x06 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_CARD_READER.

COD_MINOR_PERIPHERAL_COMBO Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_COMBO 0xC0
```

Description

This is macro COD_MINOR_PERIPHERAL_COMBO.

COD_MINOR_PERIPHERAL_DIGITIZER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_DIGITIZER (0x05 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_DIGITIZER.

COD_MINOR_PERIPHERAL_GAMEPAD Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_GAMEPAD (0x02 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_GAMEPAD.

COD_MINOR_PERIPHERAL_JOYSTICK Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_JOYSTICK (0x01 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_JOYSTICK.

COD_MINOR_PERIPHERAL_KEYBOARD Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_KEYBOARD 0x40
```

Description

This is macro COD_MINOR_PERIPHERAL_KEYBOARD.

COD_MINOR_PERIPHERAL_MOUSE Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_MOUSE 0x80
```

Description

This is macro COD_MINOR_PERIPHERAL_MOUSE.

COD_MINOR_PERIPHERAL_OTHER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_OTHER 0x00
```

Description

This is macro COD_MINOR_PERIPHERAL_OTHER.

COD_MINOR_PERIPHERAL_REMOTE Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_REMOTE (0x03 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_REMOTE.

COD_MINOR_PERIPHERAL_SENSING Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_SENSING (0x04 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_SENSING.

COD_MINOR_PERIPHERAL_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_PERIPHERAL_UNCATEGORIZED.

COD_MINOR_PHONE_CELLULAR Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_CELLULAR (0x01 << 2)
```

Description

This is macro COD_MINOR_PHONE_CELLULAR.

COD_MINOR_PHONE_CORDLESS Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_CORDLESS (0x02 << 2)
```

Description

This is macro COD_MINOR_PHONE_CORDLESS.

COD_MINOR_PHONE_ISDN Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_ISDN (0x05 << 2)
```

Description

This is macro COD_MINOR_PHONE_ISDN.

COD_MINOR_PHONE_SMART Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_SMART (0x03 << 2)
```

Description

This is macro COD_MINOR_PHONE_SMART.

COD_MINOR_PHONE_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_PHONE_UNCATEGORIZED.

COD_MINOR_PHONE_WIREDMODEM Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_WIREDMODEM (0x04 << 2)
```

Description

This is macro COD_MINOR_PHONE_WIREDMODEM.

COD_MINOR_TOY_CONTROLLER Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_CONTROLLER (0x04 << 2)
```

Description

This is macro COD_MINOR_TOY_CONTROLLER.

COD_MINOR_TOY_DOLL Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_DOLL (0x03 << 2)
```

Description

This is macro COD_MINOR_TOY_DOLL.

COD_MINOR_TOY_GAME Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_GAME (0x05 << 2)
```

Description

This is macro COD_MINOR_TOY_GAME.

COD_MINOR_TOY_ROBOT Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_ROBOT (0x01 << 2)
```

Description

This is macro COD_MINOR_TOY_ROBOT.

COD_MINOR_TOY_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_TOY_UNCATEGORIZED.

COD_MINOR_TOY_VEHICLE Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_VEHICLE (0x02 << 2)
```

Description

This is macro COD_MINOR_TOY_VEHICLE.

COD_MINOR_WEARABLE_GLASSES Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_GLASSES (0x05 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_GLASSES.

COD_MINOR_WEARABLE_HELMET Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_HELMET (0x04 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_HELMET.

COD_MINOR_WEARABLE_JACKET Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_JACKET (0x03 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_JACKET.

COD_MINOR_WEARABLE_PAGER Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_PAGER (0x02 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_PAGER.

COD_MINOR_WEARABLE_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_WEARABLE_UNCATEGORIZED.

COD_MINOR_WEARABLE_WATCH Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_WATCH (0x01 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_WATCH.

COS_AUDIO Macro

File

baseband.h

C

```
#define COS_AUDIO 0x200000
```

Description

This is macro COS_AUDIO.

COS_CAPTURING Macro

File

baseband.h

C

```
#define COS_CAPTURING 0x080000
```

Description

This is macro COS_CAPTURING.

COS_INFORMATION Macro

File

baseband.h

C

```
#define COS_INFORMATION 0x800000
```

Description

This is macro COS_INFORMATION.

COS_LIMITED_DISCOVERABLE_MODE Macro

File

baseband.h

C

```
#define COS_LIMITED_DISCOVERABLE_MODE 0x002000
```

Description

This is macro COS_LIMITED_DISCOVERABLE_MODE.

COS_NETWORKING Macro

File

baseband.h

C

```
#define COS_NETWORKING 0x020000
```

Description

This is macro COS_NETWORKING.

COS_OBJECTTRANSFER Macro

File

baseband.h

C

```
#define COS_OBJECTTRANSFER 0x100000
```

Description

This is macro COS_OBJECTTRANSFER.

COS_POSITIONING Macro

File

baseband.h

C

```
#define COS_POSITIONING 0x010000
```

Description

This is macro COS_POSITIONING.

COS_RENDERING Macro

File

baseband.h

C

```
#define COS_RENDERING 0x040000
```

Description

This is macro COS_RENDERING.

COS_TELEPHONY Macro

File

baseband.h

C

```
#define COS_TELEPHONY 0x400000
```

Description

This is macro COS_TELEPHONY.

HCI_ACCEPT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_ACCEPT_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0009)
```

Description

This is macro HCI_ACCEPT_CONNECTION_REQUEST.

HCI_ACCEPT_SYNCH_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_ACCEPT_SYNCH_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0029)
```

Description

This is macro HCI_ACCEPT_SYNCH_CONNECTION_REQUEST.

HCI_ACL_DATA_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_ACL_DATA_HEADER_LEN 4
```

Description

This is macro HCI_ACL_DATA_HEADER_LEN.

HCI_AUTHENTICATION_REQUESTED Macro

File

hci.h

C

```
#define HCI_AUTHENTICATION_REQUESTED HCI_OPCODE(OGF_LINK_CONTROL, 0x0011)
```

Description

This is macro HCI_AUTHENTICATION_REQUESTED.

HCI_BB_PACKET_TYPE_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH1 0x0010
```

Description

This is macro HCI_BB_PACKET_TYPE_DH1.

HCI_BB_PACKET_TYPE_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH3 0x0800
```

Description

This is macro HCI_BB_PACKET_TYPE_DH3.

HCI_BB_PACKET_TYPE_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH5 0x8000
```

Description

This is macro HCI_BB_PACKET_TYPE_DH5.

HCI_BB_PACKET_TYPE_DM1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM1 0x0008
```

Description

This is macro HCI_BB_PACKET_TYPE_DM1.

HCI_BB_PACKET_TYPE_DM3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM3 0x0400
```

Description

This is macro HCI_BB_PACKET_TYPE_DM3.

HCI_BB_PACKET_TYPE_DM5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM5 0x4000
```

Description

This is macro HCI_BB_PACKET_TYPE_DM5.

HCI_BB_PACKET_TYPE_NO_2_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH1 0x0002
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH1.

HCI_BB_PACKET_TYPE_NO_2_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH3 0x0100
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH3.

HCI_BB_PACKET_TYPE_NO_2_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH5 0x1000
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH5.

HCI_BB_PACKET_TYPE_NO_3_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH1 0x0004
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH1.

HCI_BB_PACKET_TYPE_NO_3_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH3 0x0200
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH3.

HCI_BB_PACKET_TYPE_NO_3_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH5 0x2000
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH5.

HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE 0x1
```

Description

This is macro HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE.

HCI_C2H_BROADCAST_P2P Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_P2P 0x0
```

Description

This is macro HCI_C2H_BROADCAST_P2P.

HCI_C2H_BROADCAST_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_PARCKED_SLAVE 0x2
```

Description

This is macro HCI_C2H_BROADCAST_PARCKED_SLAVE.

HCI_C2H_BROADCAST_RESERVED Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_RESERVED 0x3
```

Description

This is macro HCI_C2H_BROADCAST_RESERVED.

HCI_CHANGE_CONNECTION_LINK_KEY Macro

File

hci.h

C

```
#define HCI_CHANGE_CONNECTION_LINK_KEY HCI_OPCODE(OGF_LINK_CONTROL, 0x0015)
```

Description

This is macro HCI_CHANGE_CONNECTION_LINK_KEY.

HCI_CHANGE_CONNECTION_PACKET_TYPE Macro

File

hci.h

C

```
#define HCI_CHANGE_CONNECTION_PACKET_TYPE HCI_OPCODE(OGF_LINK_CONTROL, 0x000F)
```

Description

This is macro HCI_CHANGE_CONNECTION_PACKET_TYPE.

HCI_CMD_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_CMD_HEADER_LEN 3
```

Description

This is macro HCI_CMD_HEADER_LEN.

HCI_CMD_STATUS_BEING_SENT Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_BEING_SENT 2
```

Description

This is macro HCI_CMD_STATUS_BEING_SENT.

HCI_CMD_STATUS_PENDING Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_PENDING 0
```

Description

This is macro HCI_CMD_STATUS_PENDING.

HCI_CMD_STATUS_WAITING_RESPONSE Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro HCI_CMD_STATUS_WAITING_RESPONSE.

HCI_CONFIG_BECOME_MASTER Macro

File

hci.h

C

```
#define HCI_CONFIG_BECOME_MASTER 0x04
```

Description

This is macro HCI_CONFIG_BECOME_MASTER.

HCI_CONFIG_ENABLE_AUTHENTICATION Macro

File

hci.h

C

```
#define HCI_CONFIG_ENABLE_AUTHENTICATION 0x01
```

Description

This is macro HCI_CONFIG_ENABLE_AUTHENTICATION.

HCI_CONFIG_ENABLE_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_CONFIG_ENABLE_ENCRYPTION 0x02
```

Description

This is macro HCI_CONFIG_ENABLE_ENCRYPTION.

HCI_CONN_ROLE_MASTER Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_ROLE_MASTER 0
```

Description

This is macro HCI_CONN_ROLE_MASTER.

HCI_CONN_ROLE_SLAVE Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_ROLE_SLAVE 1
```

Description

This is macro HCI_CONN_ROLE_SLAVE.

HCI_CONN_STATE_AUTHENTICATING Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_AUTHENTICATING 1
```

Description

This is macro HCI_CONN_STATE_AUTHENTICATING.

HCI_CONN_STATE_CLOSED Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_CLOSED 0
```

Description

This is macro HCI_CONN_STATE_CLOSED.

HCI_CONN_STATE_OPEN Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_OPEN 2
```

Description

This is macro HCI_CONN_STATE_OPEN.

HCI_CONN_TYPE_ACL Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_ACL 1
```

Description

This is macro HCI_CONN_TYPE_ACL.

HCI_CONN_TYPE_ESCO Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_ESCO 2
```

Description

This is macro HCI_CONN_TYPE_ESCO.

HCI_CONN_TYPE_SCO Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_SCO 0
```

Description

This is macro HCI_CONN_TYPE_SCO.

HCI_CONNECTABLE Macro

File

hci.h

C

```
#define HCI_CONNECTABLE 0x02
```

Description

This is macro HCI_CONNECTABLE.

HCI_CONTROLLER Macro

File

hci.h

C

```
#define HCI_CONTROLLER _phci_ctrl
```

Description

This is macro HCI_CONTROLLER.

HCI_CREATE_CONNECTION Macro

File

hci.h

C

```
#define HCI_CREATE_CONNECTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0005)
```

Description

This is macro HCI_CREATE_CONNECTION.

HCI_CREATE_CONNECTION_CANCEL Macro

File

hci.h

C

```
#define HCI_CREATE_CONNECTION_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x0008)
```

Description

This is macro HCI_CREATE_CONNECTION_CANCEL.

HCI_CREATE_NEW_UNIT_KEY Macro

File

hci.h

C

```
#define HCI_CREATE_NEW_UNIT_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000B)
```

Description

This is macro HCI_CREATE_NEW_UNIT_KEY.

HCI_CTRL_LISTENER_ACL_DATA Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_ACL_DATA 1
```

Description

This is macro HCI_CTRL_LISTENER_ACL_DATA.

HCI_CTRL_LISTENER_EVENT Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_EVENT 0
```

Description

This is macro HCI_CTRL_LISTENER_EVENT.

HCI_CTRL_LISTENER_SCO_DATA Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_SCO_DATA 2
```

Description

This is macro HCI_CTRL_LISTENER_SCO_DATA.

HCI_CTRL_STATE_CLOSED Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_CLOSED 0x00
```

Description

This is macro HCI_CTRL_STATE_CLOSED.

HCI_CTRL_STATE_CONNECTABLE Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_CONNECTABLE 0x20
```

Description

This is macro HCI_CTRL_STATE_CONNECTABLE.

HCI_CTRL_STATE_DISCOVERABLE Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_DISCOVERABLE 0x10
```

Description

This is macro HCI_CTRL_STATE_DISCOVERABLE.

HCI_CTRL_STATE_INIT Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_INIT 0x01
```

Description

This is macro HCI_CTRL_STATE_INIT.

HCI_CTRL_STATE_READY Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_READY 0x02
```

Description

This is macro HCI_CTRL_STATE_READY.

HCI_CTRL_STATE_SLEEP Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_SLEEP 0x04
```

Description

This is macro HCI_CTRL_STATE_SLEEP.

HCI_CTRL_STATE_WAKING_UP Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_WAKING_UP 0x08
```

Description

This is macro HCI_CTRL_STATE_WAKING_UP.

HCI_DATA_BUFFER_STATE_FREE Macro

File

hci_data_buffer.h

C

```
#define HCI_DATA_BUFFER_STATE_FREE 0
```

Description

This is macro HCI_DATA_BUFFER_STATE_FREE.

HCI_DATA_BUFFER_STATE_USED Macro

File

hci_data_buffer.h

C

```
#define HCI_DATA_BUFFER_STATE_USED 1
```

Description

This is macro HCI_DATA_BUFFER_STATE_USED.

HCI_DATA_STATUS_BEING_SENT Macro

File

hci.h

C

```
#define HCI_DATA_STATUS_BEING_SENT 2
```

Description

This is macro HCI_DATA_STATUS_BEING_SENT.

HCI_DATA_STATUS_PENDING Macro

File

hci.h

C

```
#define HCI_DATA_STATUS_PENDING 0
```

Description

This is macro HCI_DATA_STATUS_PENDING.

HCI_DEFAULT_ACL_CONFIG Macro

File

hci.h

C

```
#define HCI_DEFAULT_ACL_CONFIG 0
```

Description

Default value for acl_config parameter of bt_hci_connect()

HCI_DELETE_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_DELETE_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0012)
```

Description

This is macro HCI_DELETE_STORED_LINK_KEY.

HCI_DISCONNECT Macro

File

hci.h

C

```
#define HCI_DISCONNECT HCI_OPCODE(OGF_LINK_CONTROL, 0x0006)
```

Description

This is macro HCI_DISCONNECT.

HCI_DISCOVERABLE Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE 0x01
```

Description

This is macro HCI_DISCOVERABLE.

HCI_EIR_FEC_NOT_REQUIRED Macro

File

hci_eir.h

C

```
#define HCI_EIR_FEC_NOT_REQUIRED 0
```

Description

This is macro HCI_EIR_FEC_NOT_REQUIRED.

HCI_EIR_FEC_REQUIRED Macro

File

hci_eir.h

C

```
#define HCI_EIR_FEC_REQUIRED 1
```

Description

This is macro HCI_EIR_FEC_REQUIRED.

HCI_EIR_TYPE_DEVICE_ID Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_DEVICE_ID 0x10
```

Description

This is macro HCI_EIR_TYPE_DEVICE_ID.

HCI_EIR_TYPE_FLAGS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_FLAGS 0x01
```

Description

This is macro HCI_EIR_TYPE_FLAGS.

HCI_EIR_TYPE_LOCAL_NAME_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LOCAL_NAME_COMPLETE 0x09
```

Description

This is macro HCI_EIR_TYPE_LOCAL_NAME_COMPLETE.

HCI_EIR_TYPE_LOCAL_NAME_SHORTENED Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LOCAL_NAME_SHORTENED 0x08
```

Description

This is macro HCI_EIR_TYPE_LOCAL_NAME_SHORTENED.

HCI_EIR_TYPE_MANUFACTURER_SPECIFIC Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_MANUFACTURER_SPECIFIC 0xFF
```

Description

This is macro HCI_EIR_TYPE_MANUFACTURER_SPECIFIC.

HCI_EIR_TYPE_OOB_COD Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_COD 0x0d
```

Description

This is macro HCI_EIR_TYPE_OOB_COD.

HCI_EIR_TYPE_OOB_HASH Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_HASH 0x0e
```

Description

This is macro HCI_EIR_TYPE_OOB_HASH.

HCI_EIR_TYPE_OOB_RANDOMIZER Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_RANDOMIZER 0x0f
```

Description

This is macro HCI_EIR_TYPE_OOB_RANDOMIZER.

HCI_EIR_TYPE_TX_POWER_LEVEL Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_TX_POWER_LEVEL 0x0a
```

Description

This is macro HCI_EIR_TYPE_TX_POWER_LEVEL.

HCI_EIR_TYPE_UUID128_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID128_LIST_COMPLETE 0x07
```

Description

This is macro HCI_EIR_TYPE_UUID128_LIST_COMPLETE.

HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE 0x06
```

Description

This is macro HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE.

HCI_EIR_TYPE_UUID16_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID16_LIST_COMPLETE 0x03
```

Description

This is macro HCI_EIR_TYPE_UUID16_LIST_COMPLETE.

HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE 0x02
```

Description

This is macro HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE.

HCI_EIR_TYPE_UUID32_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID32_LIST_COMPLETE 0x05
```

Description

This is macro HCI_EIR_TYPE_UUID32_LIST_COMPLETE.

HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE 0x04
```

Description

This is macro HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE.

HCI_ENABLE_DEVICE_UNDER_TEST_MODE Macro

File

hci.h

C

```
#define HCI_ENABLE_DEVICE_UNDER_TEST_MODE HCI_OPCODE(OGF_TESTING, 0x0003)
```

Description

This is macro HCI_ENABLE_DEVICE_UNDER_TEST_MODE.

HCI_ENCRYPTION_OFF Macro

File

hci.h

C

```
#define HCI_ENCRYPTION_OFF 0x00
```

Description

This is macro HCI_ENCRYPTION_OFF.

HCI_ENCRYPTION_ON Macro

File

hci.h

C

```
#define HCI_ENCRYPTION_ON 0x01
```

Description

This is macro HCI_ENCRYPTION_ON.

HCI_ENHANCED_FLUSH Macro

File

hci.h

C

```
#define HCI_ENHANCED_FLUSH HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005F)
```

Description

This is macro HCI_ENHANCED_FLUSH.

HCI_ERR_ACL_CONN_ALREADY_EXISTS Macro

File

hci_errors.h

C

```
#define HCI_ERR_ACL_CONN_ALREADY_EXISTS 0x0b
```

Description

This is macro HCI_ERR_ACL_CONN_ALREADY_EXISTS.

HCI_ERR_AUTHENTICATION_FAILURE Macro

File

hci_errors.h

C

```
#define HCI_ERR_AUTHENTICATION_FAILURE 0x05
```

Description

This is macro HCI_ERR_AUTHENTICATION_FAILURE.

HCI_ERR_CONN_REJECT_LIMITED_RESOURCES Macro

File

hci_errors.h

C

```
#define HCI_ERR_CONN_REJECT_LIMITED_RESOURCES 0x0d
```

Description

This is macro HCI_ERR_CONN_REJECT_LIMITED_RESOURCES.

HCI_ERR_INVALID_PARAMETERS Macro

File

hci_errors.h

C

```
#define HCI_ERR_INVALID_PARAMETERS 0x12
```

Description

This is macro HCI_ERR_INVALID_PARAMETERS.

HCI_ERR_MEMORY_CAPACITY_EXCEEDED Macro

File

hci_errors.h

C

```
#define HCI_ERR_MEMORY_CAPACITY_EXCEEDED 0x07
```

Description

This is macro HCI_ERR_MEMORY_CAPACITY_EXCEEDED.

HCI_ERR_SCO_CONN_LIMIT_EXCEEDED Macro

File

hci_errors.h

C

```
#define HCI_ERR_SCO_CONN_LIMIT_EXCEEDED 0x0a
```

Description

This is macro HCI_ERR_SCO_CONN_LIMIT_EXCEEDED.

HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED Macro

File

hci_errors.h

C

```
#define HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED 0x37
```

Description

This is macro HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED.

HCI_ERR_SUCCESS Macro

File

hci_errors.h

C

```
#define HCI_ERR_SUCCESS 0x00
```

Description

- addtogroup hci
- @{
- @name Errors

HCI_ERR_UNSPECIFIED Macro

File

hci_errors.h

C

```
#define HCI_ERR_UNSPECIFIED 0x1F
```

Description

This is macro HCI_ERR_UNSPECIFIED.

HCI_EVT_ALL_HCI_EVENTS Macro

File

hci.h

C

```
#define HCI_EVT_ALL_HCI_EVENTS 0xffff
```

Description

Used in bt_hci_ctrl_register_event_listener to indicate that the listener is called for all HCI events (excluding dotstack internal ones)

HCI_EVT_AUTHENTICATION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_AUTHENTICATION_COMPLETE 0x06
```

Description

This is macro HCI_EVT_AUTHENTICATION_COMPLETE.

HCI_EVT_CHANGE_CONN_LINK_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_CHANGE_CONN_LINK_COMPLETE 0x09
```

Description

This is macro HCI_EVT_CHANGE_CONN_LINK_COMPLETE.

HCI_EVT_CMD_SEND_FINISHED Macro

File

hci.h

C

```
#define HCI_EVT_CMD_SEND_FINISHED 0xff01
```

Description

This is macro HCI_EVT_CMD_SEND_FINISHED.

HCI_EVT_CMD_SEND_STARTED Macro

File

hci.h

C

```
#define HCI_EVT_CMD_SEND_STARTED 0xff00
```

Description

dotstack internal events

HCI_EVT_COMMAND_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_COMPLETE 0x0E
```

Description

This is macro HCI_EVT_COMMAND_COMPLETE.

HCI_EVT_COMMAND_COMPLETE_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_COMPLETE_PARAM_LEN 3
```

Description

This is macro HCI_EVT_COMMAND_COMPLETE_PARAM_LEN.

HCI_EVT_COMMAND_STATUS Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_STATUS 0x0F
```

Description

This is macro HCI_EVT_COMMAND_STATUS.

HCI_EVT_CONN_PACKET_TYPE_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_CONN_PACKET_TYPE_CHANGED 0x1D
```

Description

This is macro HCI_EVT_CONN_PACKET_TYPE_CHANGED.

HCI_EVT_CONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_CONNECTION_COMPLETE 0x03
```

Description

This is macro HCI_EVT_CONNECTION_COMPLETE.

HCI_EVT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_CONNECTION_REQUEST 0x04
```

Description

This is macro HCI_EVT_CONNECTION_REQUEST.

HCI_EVT_DATA_BUFFER_OVERFLOW Macro

File

hci.h

C

```
#define HCI_EVT_DATA_BUFFER_OVERFLOW 0x1A
```

Description

This is macro HCI_EVT_DATA_BUFFER_OVERFLOW.

HCI_EVT_DISCONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_DISCONNECTION_COMPLETE 0x05
```

Description

This is macro HCI_EVT_DISCONNECTION_COMPLETE.

HCI_EVT_ENCRYPTION_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_ENCRYPTION_CHANGE 0x08
```

Description

This is macro HCI_EVT_ENCRYPTION_CHANGE.

HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE 0x30
```

Description

This is macro HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE.

HCI_EVT_ENHANCED_FLUSH_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_ENHANCED_FLUSH_COMPLETE 0x39
```

Description

This is macro HCI_EVT_ENHANCED_FLUSH_COMPLETE.

HCI_EVT_EXTENDED_INQUIRY_RESULT Macro

File

hci.h

C

```
#define HCI_EVT_EXTENDED_INQUIRY_RESULT 0x2F
```

Description

This is macro HCI_EVT_EXTENDED_INQUIRY_RESULT.

HCI_EVT_FIRST Macro

File

hci.h

C

```
#define HCI_EVT_FIRST HCI_EVT_INQUIRY_COMPLETE
```

Description

This is macro HCI_EVT_FIRST.

HCI_EVT_FLOW_SPECIFICATION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_FLOW_SPECIFICATION_COMPLETE 0x21
```

Description

This is macro HCI_EVT_FLOW_SPECIFICATION_COMPLETE.

HCI_EVT_FLUSH_OCCURED Macro

File

hci.h

C

```
#define HCI_EVT_FLUSH_OCCURED 0x11
```

Description

This is macro HCI_EVT_FLUSH_OCCURED.

HCI_EVT_HARDWARE_ERROR Macro

File

hci.h

C

```
#define HCI_EVT_HARDWARE_ERROR 0x10
```

Description

This is macro HCI_EVT_HARDWARE_ERROR.

HCI_EVT_INQUIRY_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_COMPLETE 0x01
```

Description

- addtogroup hci
- @{
- @name Events

HCI_EVT_INQUIRY_RESULT Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_RESULT 0x02
```

Description

This is macro HCI_EVT_INQUIRY_RESULT.

HCI_EVT_INQUIRY_RESULT_WITH_RSSI Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_RESULT_WITH_RSSI 0x22
```

Description

This is macro HCI_EVT_INQUIRY_RESULT_WITH_RSSI.

HCI_EVT_IO_CAPABILITY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_IO_CAPABILITY_REQUEST 0x31
```

Description

This is macro HCI_EVT_IO_CAPABILITY_REQUEST.

HCI_EVT_IO_CAPABILITY_RESPONSE Macro

File

hci.h

C

```
#define HCI_EVT_IO_CAPABILITY_RESPONSE 0x32
```

Description

This is macro HCI_EVT_IO_CAPABILITY_RESPONSE.

HCI_EVT_KEYPRESS_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_KEYPRESS_NOTIFICATION 0x3C
```

Description

This is macro HCI_EVT_KEYPRESS_NOTIFICATION.

HCI_EVT_LAST Macro

File

hci.h

C

```
#define HCI_EVT_LAST HCI_EVT_LE_META_EVENT
```

Description

This is macro HCI_EVT_LAST.

HCI_EVT_LE_META_EVENT Macro

File

hci.h

C

```
#define HCI_EVT_LE_META_EVENT 0x3E
```

Description

This is macro HCI_EVT_LE_META_EVENT.

HCI_EVT_LINK_IS_BUSY Macro

File

hci.h

C

```
#define HCI_EVT_LINK_IS_BUSY 0xff03
```

Description

This is macro HCI_EVT_LINK_IS_BUSY.

HCI_EVT_LINK_IS_IDLE Macro

File

hci.h

C

```
#define HCI_EVT_LINK_IS_IDLE 0xff02
```

Description

This is macro HCI_EVT_LINK_IS_IDLE.

HCI_EVT_LINK_KEY_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_LINK_KEY_NOTIFICATION 0x18
```

Description

This is macro HCI_EVT_LINK_KEY_NOTIFICATION.

HCI_EVT_LINK_KEY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_LINK_KEY_REQUEST 0x17
```

Description

This is macro HCI_EVT_LINK_KEY_REQUEST.

HCI_EVT_LINK_SUPERVISION_TO_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_LINK_SUPERVISION_TO_CHANGED 0x38
```

Description

This is macro HCI_EVT_LINK_SUPERVISION_TO_CHANGED.

HCI_EVT_LOOPBACK_COMMAND Macro

File

hci.h

C

```
#define HCI_EVT_LOOPBACK_COMMAND 0x19
```

Description

This is macro HCI_EVT_LOOPBACK_COMMAND.

HCI_EVT_MASTER_LINK_KEY_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_MASTER_LINK_KEY_COMPLETE 0x0A
```

Description

This is macro HCI_EVT_MASTER_LINK_KEY_COMPLETE.

HCI_EVT_MAX_SLOTS_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_MAX_SLOTS_CHANGE 0x1B
```

Description

This is macro HCI_EVT_MAX_SLOTS_CHANGE.

HCI_EVT_MODE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_MODE_CHANGE 0x14
```

Description

This is macro HCI_EVT_MODE_CHANGE.

HCI_EVT_NUM_OF_COMPLETED_PACKETS Macro

File

hci.h

C

```
#define HCI_EVT_NUM_OF_COMPLETED_PACKETS 0x13
```

Description

This is macro HCI_EVT_NUM_OF_COMPLETED_PACKETS.

HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE 0x20
```

Description

This is macro HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE.

HCI_EVT_PIN_CODE_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_PIN_CODE_REQUEST 0x16
```

Description

This is macro HCI_EVT_PIN_CODE_REQUEST.

HCI_EVT_QOS_SETUP_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_QOS_SETUP_COMPLETE 0x0D
```

Description

This is macro HCI_EVT_QOS_SETUP_COMPLETE.

HCI_EVT_QOS_VIOLATION Macro

File

hci.h

C

```
#define HCI_EVT_QOS_VIOLATION 0x1E
```

Description

This is macro HCI_EVT_QOS_VIOLATION.

HCI_EVT_READ_CLOCK_OFFSET_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_READ_CLOCK_OFFSET_COMPLETE 0x1C
```

Description

This is macro HCI_EVT_READ_CLOCK_OFFSET_COMPLETE.

HCI_EVT_READ_RMT_EXT_FEATURES_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_EXT_FEATURES_COMP 0x23
```

Description

This is macro HCI_EVT_READ_RMT_EXT_FEATURES_COMP.

HCI_EVT_READ_RMT_SUP_FEATURES_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_SUP_FEATURES_COMP 0x0B
```

Description

This is macro HCI_EVT_READ_RMT_SUP_FEATURES_COMP.

HCI_EVT_READ_RMT_VERSION_INFO_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_VERSION_INFO_COMP 0x0C
```

Description

This is macro HCI_EVT_READ_RMT_VERSION_INFO_COMP.

HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE 0x07
```

Description

This is macro HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE.

HCI_EVT_REMOTE_OOB_DATA_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_REMOTE_OOB_DATA_REQUEST 0x35
```

Description

This is macro HCI_EVT_REMOTE_OOB_DATA_REQUEST.

HCI_EVT_RETURN_LINK_KEYS Macro

File

hci.h

C

```
#define HCI_EVT_RETURN_LINK_KEYS 0x15
```

Description

This is macro HCI_EVT_RETURN_LINK_KEYS.

HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF Macro

File

hci.h

C

```
#define HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF 0x3D
```

Description

This is macro HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF.

HCI_EVT_ROLE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_ROLE_CHANGE 0x12
```

Description

This is macro HCI_EVT_ROLE_CHANGE.

HCI_EVT_SIMPLE_PAIRING_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_SIMPLE_PAIRING_COMPLETE 0x36
```

Description

This is macro HCI_EVT_SIMPLE_PAIRING_COMPLETE.

HCI_EVT_SNIFF_SUBRATING Macro

File

hci.h

C

```
#define HCI_EVT_SNIFF_SUBRATING 0x2E
```

Description

This is macro HCI_EVT_SNIFF_SUBRATING.

HCI_EVT_SYNCH_CONNECTION_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_SYNCH_CONNECTION_CHANGED 0x2D
```

Description

This is macro HCI_EVT_SYNCH_CONNECTION_CHANGED.

HCI_EVT_SYNCH_CONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_SYNCH_CONNECTION_COMPLETE 0x2C
```

Description

This is macro HCI_EVT_SYNCH_CONNECTION_COMPLETE.

HCI_EVT_USER_CONFIRMATION_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_USER_CONFIRMATION_REQUEST 0x33
```

Description

This is macro HCI_EVT_USER_CONFIRMATION_REQUEST.

HCI_EVT_USER_PASSKEY_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_USER_PASSKEY_NOTIFICATION 0x3B
```

Description

This is macro HCI_EVT_USER_PASSKEY_NOTIFICATION.

HCI_EVT_USER_PASSKEY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_USER_PASSKEY_REQUEST 0x34
```

Description

This is macro HCI_EVT_USER_PASSKEY_REQUEST.

HCI_EXIT_PARK_STATE Macro

File

hci.h

C

```
#define HCI_EXIT_PARK_STATE HCI_OPCODE(OGF_LINK_POLICY, 0x0006)
```

Description

This is macro HCI_EXIT_PARK_STATE.

HCI_EXIT_PERIODIC_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_EXIT_PERIODIC_INQUIRY_MODE HCI_OPCODE(OGF_LINK_CONTROL, 0x0004)
```

Description

This is macro HCI_EXIT_PERIODIC_INQUIRY_MODE.

HCI_EXIT_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_EXIT_SNIFF_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0004)
```

Description

This is macro HCI_EXIT_SNIFF_MODE.

HCI_FLOW_SPECIFICATION Macro

File

hci.h

C

```
#define HCI_FLOW_SPECIFICATION HCI_OPCODE(OGF_LINK_POLICY, 0x0010)
```

Description

This is macro HCI_FLOW_SPECIFICATION.

HCI_FLUSH Macro

File

hci.h

C

```
#define HCI_FLUSH HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0008)
```

Description

This is macro HCI_FLUSH.

HCI_H2C_BROADCAST_ACTIVE_SLAVE Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_ACTIVE_SLAVE 0x1
```

Description

This is macro HCI_H2C_BROADCAST_ACTIVE_SLAVE.

HCI_H2C_BROADCAST_NO_BROADCATS Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_NO_BROADCATS 0x0
```

Description

This is macro HCI_H2C_BROADCAST_NO_BROADCATS.

HCI_H2C_BROADCAST_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_PARCKED_SLAVE 0x2
```

Description

This is macro HCI_H2C_BROADCAST_PARCKED_SLAVE.

HCI_H2C_BROADCAST_RESERVED Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_RESERVED 0x3
```

Description

This is macro HCI_H2C_BROADCAST_RESERVED.

HCI_HOLD_MODE Macro

File

hci.h

C

```
#define HCI_HOLD_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0001)
```

Description

addtogroup hci @{

@name Link policy commands

details The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.

HCI_HOST_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_HOST_BUFFER_SIZE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0033)
```

Description

This is macro HCI_HOST_BUFFER_SIZE.

HCI_HOST_NUM_OF_COMPLETED_PACKETS Macro

File

hci.h

C

```
#define HCI_HOST_NUM_OF_COMPLETED_PACKETS HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0035)
```

Description

This is macro HCI_HOST_NUM_OF_COMPLETED_PACKETS.

HCI_INQUIRY Macro

File

hci.h

C

```
#define HCI_INQUIRY HCI_OPCODE(OGF_LINK_CONTROL, 0x0001)
```

Description

addtogroup hci @{
@name Link control commands
details The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.

HCI_INQUIRY_CANCEL Macro

File

hci.h

C

```
#define HCI_INQUIRY_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x0002)
```

Description

This is macro HCI_INQUIRY_CANCEL.

HCI_INQUIRY_MODE_EXTENDED Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_EXTENDED 2
```

Description

This is macro HCI_INQUIRY_MODE_EXTENDED.

HCI_INQUIRY_MODE_STANDARD Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_STANDARD 0
```

Description

This is macro HCI_INQUIRY_MODE_STANDARD.

HCI_INQUIRY_MODE_WITH_RSSI Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_WITH_RSSI 1
```

Description

This is macro HCI_INQUIRY_MODE_WITH_RSSI.

HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0034)
```

Description

This is macro HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY.

HCI_IO_CAPABILITY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_IO_CAPABILITY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002B)
```

Description

This is macro HCI_IO_CAPABILITY_REQUEST_REPLY.

HCI_LE_ADD_DEVICE_TO_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_ADD_DEVICE_TO_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0011)
```

Description

This is macro HCI_LE_ADD_DEVICE_TO_WHITE_LIST.

HCI_LE_ADDRESS_TYPE_PUBLIC Macro

File

hci_le.h

C

```
#define HCI_LE_ADDRESS_TYPE_PUBLIC 0
```

Description

This is macro HCI_LE_ADDRESS_TYPE_PUBLIC.

HCI_LE_ADDRESS_TYPE_RANDOM Macro

File

hci_le.h

C

```
#define HCI_LE_ADDRESS_TYPE_RANDOM 1
```

Description

This is macro HCI_LE_ADDRESS_TYPE_RANDOM.

HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED 0x04
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED.

HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE 0x02
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE.

HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE 0x01
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE.

HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER 0x08
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER.

HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST 0x10
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST.

HCI_LE_AVERTISING_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_AVERTISING_DISABLED 0
```

Description

This is macro HCI_LE_AVERTISING_DISABLED.

HCI_LE_AVERTISING_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_AVERTISING_ENABLED 1
```

Description

This is macro HCI_LE_AVERTISING_ENABLED.

HCI_LE_CLEAR_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_CLEAR_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0010)
```

Description

This is macro HCI_LE_CLEAR_WHITE_LIST.

HCI_LE_CONNECTION_UPDATE Macro

File

hci.h

C

```
#define HCI_LE_CONNECTION_UPDATE HCI_OPCODE(OGF_LE, 0x0013)
```

Description

This is macro HCI_LE_CONNECTION_UPDATE.

HCI_LE_CREATE_CONNECTION Macro

File

hci.h

C

```
#define HCI_LE_CREATE_CONNECTION HCI_OPCODE(OGF_LE, 0x000D)
```

Description

This is macro HCI_LE_CREATE_CONNECTION.

HCI_LE_CREATE_CONNECTION_CANCEL Macro

File

hci.h

C

```
#define HCI_LE_CREATE_CONNECTION_CANCEL HCI_OPCODE(OGF_LE, 0x000E)
```

Description

This is macro HCI_LE_CREATE_CONNECTION_CANCEL.

HCI_LE_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DISABLED 0
```

Description

This is macro HCI_LE_DISABLED.

HCI_LE_DUPLICATE_FILTERING_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DUPLICATE_FILTERING_DISABLED 0
```

Description

This is macro HCI_LE_DUPLICATE_FILTERING_DISABLED.

HCI_LE_DUPLICATE_FILTERING_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DUPLICATE_FILTERING_ENABLED 1
```

Description

This is macro HCI_LE_DUPLICATE_FILTERING_ENABLED.

HCI_LE_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_ENABLED 1
```

Description

This is macro HCI_LE_ENABLED.

HCI_LE_ENCRYPT Macro

File

hci.h

C

```
#define HCI_LE_ENCRYPT HCI_OPCODE(OGF_LE, 0x0017)
```

Description

This is macro HCI_LE_ENCRYPT.

HCI_LE_EVT_ADVERTISING_REPORT Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_ADVERTISING_REPORT 0x02
```

Description

This is macro HCI_LE_EVT_ADVERTISING_REPORT.

HCI_LE_EVT_CONNECTION_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_CONNECTION_COMPLETE 0x01
```

Description

This is macro HCI_LE_EVT_CONNECTION_COMPLETE.

HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE 0x03
```

Description

This is macro HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE.

HCI_LE_EVT_LONG_TERM_KEY_REQUEST Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_LONG_TERM_KEY_REQUEST 0x05
```

Description

This is macro HCI_LE_EVT_LONG_TERM_KEY_REQUEST.

HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE 0x04
```

Description

This is macro HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE.

HCI_LE_FILTER_POLICY_NOT_USED Macro

File

hci_le.h

C

```
#define HCI_LE_FILTER_POLICY_NOT_USED 0 // White list is not used to determine which advertiser to connect
to.
```

Description

White list is not used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be used.

HCI_LE_FILTER_POLICY_WHITE_LIST Macro

File

hci_le.h

C

```
#define HCI_LE_FILTER_POLICY_WHITE_LIST 1 // White list is used to determine which advertiser to connect to.
```

Description

White list is used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be ignored

HCI_LE_FLAG_SHARED_ACL_BUFFERS Macro

File

hci_le.h

C

```
#define HCI_LE_FLAG_SHARED_ACL_BUFFERS 2
```

Description

This is macro HCI_LE_FLAG_SHARED_ACL_BUFFERS.

HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR Macro

File

hci_le.h

C

```
#define HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR 1
```

Description

This is macro HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR.

HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LE, 0x001B)
```

Description

This is macro HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY.

HCI_LE_LONG_TERM_KEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_LE_LONG_TERM_KEY_REQUEST_REPLY HCI_OPCODE(OGF_LE, 0x001A)
```

Description

This is macro HCI_LE_LONG_TERM_KEY_REQUEST_REPLY.

HCI_LE_MAX_AD_LEN Macro

File

hci.h

C

```
#define HCI_LE_MAX_AD_LEN 31
```

Description

This is macro HCI_LE_MAX_AD_LEN.

HCI_LE_RAND Macro

File

hci.h

C

```
#define HCI_LE_RAND HCI_OPCODE(OGF_LE, 0x0018)
```

Description

This is macro HCI_LE_RAND.

HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER Macro

File

hci.h

C

```
#define HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER HCI_OPCODE(OGF_LE, 0x0007)
```

Description

This is macro HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER.

HCI_LE_READ_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_LE_READ_BUFFER_SIZE HCI_OPCODE(OGF_LE, 0x0002)
```

Description

This is macro HCI_LE_READ_BUFFER_SIZE.

HCI_LE_READ_CHANNEL_MAP Macro

File

hci.h

C

```
#define HCI_LE_READ_CHANNEL_MAP HCI_OPCODE(OGF_LE, 0x0015)
```

Description

This is macro HCI_LE_READ_CHANNEL_MAP.

HCI_LE_READ_LOCAL_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_LE_READ_LOCAL_SUPPORTED_FEATURES HCI_OPCODE(OGF_LE, 0x0003)
```

Description

This is macro HCI_LE_READ_LOCAL_SUPPORTED_FEATURES.

HCI_LE_READ_REMOTE_USED_FEATURES Macro

File

hci.h

C

```
#define HCI_LE_READ_REMOTE_USED_FEATURES HCI_OPCODE(OGF_LE, 0x0016)
```

Description

This is macro HCI_LE_READ_REMOTE_USED_FEATURES.

HCI_LE_READ_SUPPORTED_STATES Macro

File

hci.h

C

```
#define HCI_LE_READ_SUPPORTED_STATES HCI_OPCODE(OGF_LE, 0x001C)
```

Description

This is macro HCI_LE_READ_SUPPORTED_STATES.

HCI_LE_READ_WHITE_LIST_SIZE Macro

File

hci.h

C

```
#define HCI_LE_READ_WHITE_LIST_SIZE HCI_OPCODE(OGF_LE, 0x000F)
```

Description

This is macro HCI_LE_READ_WHITE_LIST_SIZE.

HCI_LE_RECEIVE_TEST Macro

File

hci.h

C

```
#define HCI_LE_RECEIVE_TEST HCI_OPCODE(OGF_LE, 0x001D)
```

Description

This is macro HCI_LE_RECEIVE_TEST.

HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0012)
```

Description

This is macro HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST.

HCI_LE_SCAN_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_DISABLED 0
```

Description

This is macro HCI_LE_SCAN_DISABLED.

HCI_LE_SCAN_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_ENABLED 1
```

Description

This is macro HCI_LE_SCAN_ENABLED.

HCI_LE_SCAN_EVT_CANCEL_FAILED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_CANCEL_FAILED 4
```

Description

This is macro HCI_LE_SCAN_EVT_CANCEL_FAILED.

HCI_LE_SCAN_EVT_CANCELLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_CANCELLED 1
```

Description

This is macro HCI_LE_SCAN_EVT_CANCELLED.

HCI_LE_SCAN_EVT_DEVICE_FOUND Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_DEVICE_FOUND 2
```

Description

This is macro HCI_LE_SCAN_EVT_DEVICE_FOUND.

HCI_LE_SCAN_EVT_START_FAILED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_START_FAILED 3
```

Description

This is macro HCI_LE_SCAN_EVT_START_FAILED.

HCI_LE_SCAN_EVT_STARTED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_STARTED 0
```

Description

This is macro HCI_LE_SCAN_EVT_STARTED.

HCI_LE_SCAN_FILTER_POLICY_ALL Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_FILTER_POLICY_ALL 0
```

Description

This is macro HCI_LE_SCAN_FILTER_POLICY_ALL.

HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST 1
```

Description

This is macro HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST.

HCI_LE_SCAN_TYPE_ACTIVE Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_TYPE_ACTIVE 1
```

Description

This is macro HCI_LE_SCAN_TYPE_ACTIVE.

HCI_LE_SCAN_TYPE_PASSIVE Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_TYPE_PASSIVE 0
```

Description

This is macro HCI_LE_SCAN_TYPE_PASSIVE.

HCI_LE_SET_ADVERTISE_ENABLE Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISE_ENABLE HCI_OPCODE(OGF_LE, 0x000A)
```

Description

This is macro HCI_LE_SET_ADVERTISE_ENABLE.

HCI_LE_SET_ADVERTISING_DATA Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISING_DATA HCI_OPCODE(OGF_LE, 0x0008)
```

Description

This is macro HCI_LE_SET_ADVERTISING_DATA.

HCI_LE_SET_ADVERTISING_PARAMETERS Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISING_PARAMETERS HCI_OPCODE(OGF_LE, 0x0006)
```

Description

This is macro HCI_LE_SET_ADVERTISING_PARAMETERS.

HCI_LE_SET_EVENT_MASK Macro

File

hci.h

C

```
#define HCI_LE_SET_EVENT_MASK HCI_OPCODE(OGF_LE, 0x0001)
```

Description

addtogroup hci @{

@name LE controller commands

details The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.

HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION Macro

File

hci.h

C

```
#define HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION HCI_OPCODE(OGF_LE, 0x0014)
```

Description

This is macro HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION.

HCI_LE_SET_RANDOM_ADDRESS Macro

File

hci.h

C

```
#define HCI_LE_SET_RANDOM_ADDRESS HCI_OPCODE(OGF_LE, 0x0005)
```

Description

This is macro HCI_LE_SET_RANDOM_ADDRESS.

HCI_LE_SET_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_ENABLE HCI_OPCODE(OGF_LE, 0x000C)
```

Description

This is macro HCI_LE_SET_SCAN_ENABLE.

HCI_LE_SET_SCAN_PARAMETERS Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_PARAMETERS HCI_OPCODE(OGF_LE, 0x000B)
```

Description

This is macro HCI_LE_SET_SCAN_PARAMETERS.

HCI_LE_SET_SCAN_RESPONSE_DATA Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_RESPONSE_DATA HCI_OPCODE(OGF_LE, 0x0009)
```

Description

This is macro HCI_LE_SET_SCAN_RESPONSE_DATA.

HCI_LE_SIMULTANEOUS_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SIMULTANEOUS_DISABLED 0
```

Description

This is macro HCI_LE_SIMULTANEOUS_DISABLED.

HCI_LE_SIMULTANEOUS_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SIMULTANEOUS_ENABLED 1
```

Description

This is macro HCI_LE_SIMULTANEOUS_ENABLED.

HCI_LE_START_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_LE_START_ENCRYPTION HCI_OPCODE(OGF_LE, 0x0019)
```

Description

This is macro HCI_LE_START_ENCRYPTION.

HCI_LE_TEST_END Macro

File

hci.h

C

```
#define HCI_LE_TEST_END HCI_OPCODE(OGF_LE, 0x001F)
```

Description

This is macro HCI_LE_TEST_END.

HCI_LE_TRANSMITTER_TEST Macro

File

hci.h

C

```
#define HCI_LE_TRANSMITTER_TEST HCI_OPCODE(OGF_LE, 0x001E)
```

Description

This is macro HCI_LE_TRANSMITTER_TEST.

HCI_LINK_KEY_LEN Macro

File

hci.h

C

```
#define HCI_LINK_KEY_LEN 16
```

Description

This is macro HCI_LINK_KEY_LEN.

HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000C)
```

Description

This is macro HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY.

HCI_LINK_KEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_LINK_KEY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000B)
```

Description

This is macro HCI_LINK_KEY_REQUEST_REPLY.

HCI_LINK_KEY_TYPE_COMBINATION Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_COMBINATION 0x00
```

Description

This is macro HCI_LINK_KEY_TYPE_COMBINATION.

HCI_LINK_KEY_TYPE_LOCAL_UNIT Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_LOCAL_UNIT 0x01
```

Description

This is macro HCI_LINK_KEY_TYPE_LOCAL_UNIT.

HCI_LINK_KEY_TYPE_REMOTE_UNIT Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_REMOTE_UNIT 0x02
```

Description

This is macro HCI_LINK_KEY_TYPE_REMOTE_UNIT.

HCI_LINK_POLICY_ENABLE_HOLD_MODE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_HOLD_MODE 0x0002
```

Description

This is macro HCI_LINK_POLICY_ENABLE_HOLD_MODE.

HCI_LINK_POLICY_ENABLE_PARK_STATE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_PARK_STATE 0x0008
```

Description

This is macro HCI_LINK_POLICY_ENABLE_PARK_STATE.

HCI_LINK_POLICY_ENABLE_ROLE_SWITCH Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_ROLE_SWITCH 0x0001
```

Description

This is macro HCI_LINK_POLICY_ENABLE_ROLE_SWITCH.

HCI_LINK_POLICY_ENABLE_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_SNIFF_MODE 0x0004
```

Description

This is macro HCI_LINK_POLICY_ENABLE_SNIFF_MODE.

HCI_LINK_TYPE_BD_EDR Macro

File

hci_conn_state.h

C

```
#define HCI_LINK_TYPE_BD_EDR 0
```

Description

This is macro HCI_LINK_TYPE_BD_EDR.

HCI_LINK_TYPE_LE Macro

File

hci_conn_state.h

C

```
#define HCI_LINK_TYPE_LE 1
```

Description

This is macro HCI_LINK_TYPE_LE.

HCI_MASTER_LINK_KEY Macro

File

hci.h

C

```
#define HCI_MASTER_LINK_KEY HCI_OPCODE(OGF_LINK_CONTROL, 0x0017)
```

Description

This is macro HCI_MASTER_LINK_KEY.

HCI_MAX_DATA_BUFFERS Macro

File

hci_data_buffer.h

C

```
#define HCI_MAX_DATA_BUFFERS 2
```

Description

This is macro HCI_MAX_DATA_BUFFERS.

HCI_MAX_EVENT_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_MAX_EVENT_PARAM_LEN 255
```

Description

This is macro HCI_MAX_EVENT_PARAM_LEN.

HCI_MAX_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_MAX_PARAM_LEN 248
```

Description

This is macro HCI_MAX_PARAM_LEN.

HCI_MAX_PIN_LENGTH Macro

File

hci.h

C

```
#define HCI_MAX_PIN_LENGTH 8
```

Description

This is macro HCI_MAX_PIN_LENGTH.

HCI_OPCODE Macro

File

hci.h

C

```
#define HCI_OPCODE(ogf, ocf) (((int)ogf << 10) | ((int)ocf & 0x3ff))
```

Description

This is macro HCI_OPCODE.

HCI_PACKET_BOUNDARY_CONTINUE Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_CONTINUE 0x1
```

Description

This is macro HCI_PACKET_BOUNDARY_CONTINUE.

HCI_PACKET_BOUNDARY_FIRST Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST 0x2
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST.

HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH 0x2
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH.

HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH 0x0
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH.

HCI_PACKET_TYPE_ACL_DATA Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_ACL_DATA 0x02
```

Description

This is macro HCI_PACKET_TYPE_ACL_DATA.

HCI_PACKET_TYPE_COMMAND Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_COMMAND 0x01
```

Description

This is macro HCI_PACKET_TYPE_COMMAND.

HCI_PACKET_TYPE_EVENT Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_EVENT 0x04
```

Description

This is macro HCI_PACKET_TYPE_EVENT.

HCI_PACKET_TYPE_NONE Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_NONE 0x00
```

Description

This is macro HCI_PACKET_TYPE_NONE.

HCI_PACKET_TYPE_SCO_DATA Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_SCO_DATA 0x03
```

Description

This is macro HCI_PACKET_TYPE_SCO_DATA.

HCI_PAGE_SCAN_REPETITION_MODE_R0 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R0 0x00
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R0.

HCI_PAGE_SCAN_REPETITION_MODE_R1 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R1 0x01
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R1.

HCI_PAGE_SCAN_REPETITION_MODE_R2 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R2 0x02
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R2.

HCI_PARAM_LEN_BD_ADDR Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_BD_ADDR 6
```

Description

This is macro HCI_PARAM_LEN_BD_ADDR.

HCI_PARAM_LEN_BYTE Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_BYTE 1
```

Description

This is macro HCI_PARAM_LEN_BYTE.

HCI_PARAM_LEN_DEV_CLASS Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_DEV_CLASS 3
```

Description

This is macro HCI_PARAM_LEN_DEV_CLASS.

HCI_PARAM_LEN_HANDLE Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_HANDLE 4
```

Description

This is macro HCI_PARAM_LEN_HANDLE.

HCI_PARAM_LEN_INT Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_INT 2
```

Description

This is macro HCI_PARAM_LEN_INT.

HCI_PARAM_LEN_LONG Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_LONG 4
```

Description

This is macro HCI_PARAM_LEN_LONG.

HCI_PARAM_TYPE_BD_ADDR Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_BD_ADDR 4
```

Description

This is macro HCI_PARAM_TYPE_BD_ADDR.

HCI_PARAM_TYPE_BYTE Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_BYTE 0
```

Description

This is macro HCI_PARAM_TYPE_BYTE.

HCI_PARAM_TYPE_DEV_CLASS Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_DEV_CLASS 6
```

Description

This is macro HCI_PARAM_TYPE_DEV_CLASS.

HCI_PARAM_TYPE_HANDLE Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_HANDLE 5
```

Description

This is macro HCI_PARAM_TYPE_HANDLE.

HCI_PARAM_TYPE_INT Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_INT 1
```

Description

This is macro HCI_PARAM_TYPE_INT.

HCI_PARAM_TYPE_LONG Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_LONG 2
```

Description

This is macro HCI_PARAM_TYPE_LONG.

HCI_PARAM_TYPE_STRING Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_STRING 7
```

Description

This is macro HCI_PARAM_TYPE_STRING.

HCI_PARK_STATE Macro

File

hci.h

C

```
#define HCI_PARK_STATE HCI_OPCODE(OGF_LINK_POLICY, 0x0005)
```

Description

This is macro HCI_PARK_STATE.

HCI_PERIODIC_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_PERIODIC_INQUIRY_MODE HCI_OPCODE(OGF_LINK_CONTROL, 0x0003)
```

Description

This is macro HCI_PERIODIC_INQUIRY_MODE.

HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000E)
```

Description

This is macro HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY.

HCI_PIN_CODE_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_PIN_CODE_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000D)
```

Description

This is macro HCI_PIN_CODE_REQUEST_REPLY.

HCI_POWER_MODE_ACTIVE Macro

File

hci.h

C

```
#define HCI_POWER_MODE_ACTIVE 0
```

Description

This is macro HCI_POWER_MODE_ACTIVE.

HCI_POWER_MODE_HOLD Macro

File

hci.h

C

```
#define HCI_POWER_MODE_HOLD 1
```

Description

This is macro HCI_POWER_MODE_HOLD.

HCI_POWER_MODE_PARK Macro

File

hci.h

C

```
#define HCI_POWER_MODE_PARK 3
```

Description

This is macro HCI_POWER_MODE_PARK.

HCI_POWER_MODE_SNIFF Macro

File

hci.h

C

```
#define HCI_POWER_MODE_SNIFF 2
```

Description

This is macro HCI_POWER_MODE_SNIFF.

HCI_QOS_SETUP Macro

File

hci.h

C

```
#define HCI_QOS_SETUP HCI_OPCODE(OGF_LINK_POLICY, 0x0007)
```

Description

This is macro HCI_QOS_SETUP.

HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE Macro

File

hci.h

C

```
#define HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0048)
```

Description

This is macro HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE.

HCI_READ_AFH_CHANNEL_MAP Macro

File

hci.h

C

```
#define HCI_READ_AFH_CHANNEL_MAP HCI_OPCODE(OGF_STATUS, 0x0006)
```

Description

This is macro HCI_READ_AFH_CHANNEL_MAP.

HCI_READ_AUTHENTICATION_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_AUTHENTICATION_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001F)
```

Description

This is macro HCI_READ_AUTHENTICATION_ENABLE.

HCI_READ_AUTOMATIC_FLASH_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_AUTOMATIC_FLASH_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0027)
```

Description

This is macro HCI_READ_AUTOMATIC_FLASH_TIMEOUT.

HCI_READ_BD_ADDR Macro

File

hci.h

C

```
#define HCI_READ_BD_ADDR HCI_OPCODE(OGF_INFORMATION, 0x0009)
```

Description

This is macro HCI_READ_BD_ADDR.

HCI_READ_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_READ_BUFFER_SIZE HCI_OPCODE(OGF_INFORMATION, 0x0005)
```

Description

This is macro HCI_READ_BUFFER_SIZE.

HCI_READ_CLASS_OF_DEVICE Macro

File

hci.h

C

```
#define HCI_READ_CLASS_OF_DEVICE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0023)
```

Description

This is macro HCI_READ_CLASS_OF_DEVICE.

HCI_READ_CLOCK_COMMAND Macro

File

hci.h

C

```
#define HCI_READ_CLOCK_COMMAND HCI_OPCODE(OGF_STATUS, 0x0007)
```

Description

This is macro HCI_READ_CLOCK_COMMAND.

HCI_READ_CLOCK_OFFSET Macro

File

hci.h

C

```
#define HCI_READ_CLOCK_OFFSET HCI_OPCODE(OGF_LINK_CONTROL, 0x001F)
```

Description

This is macro HCI_READ_CLOCK_OFFSET.

HCI_READ_CONNECTION_ACCEPT_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_CONNECTION_ACCEPT_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0015)
```

Description

This is macro HCI_READ_CONNECTION_ACCEPT_TIMEOUT.

HCI_READ_CURRENT_IAC_LAP Macro

File

hci.h

C

```
#define HCI_READ_CURRENT_IAC_LAP HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0039)
```

Description

This is macro HCI_READ_CURRENT_IAC_LAP.

HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING Macro

File

hci.h

C

```
#define HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005A)
```

Description

This is macro HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING.

HCI_READ_DEFAULT_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_READ_DEFAULT_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000E)
```

Description

This is macro HCI_READ_DEFAULT_POLICY_SETTINGS.

HCI_READ_ENCRYPTION_MODE Macro

File

hci.h

C

```
#define HCI_READ_ENCRYPTION_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0021)
```

Description

This is macro HCI_READ_ENCRYPTION_MODE.

HCI_READ_EXTENDED_INQUIRY_RESPONSE Macro

File

hci.h

C

```
#define HCI_READ_EXTENDED_INQUIRY_RESPONSE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0051)
```

Description

This is macro HCI_READ_EXTENDED_INQUIRY_RESPONSE.

HCI_READ_FAILED_CONTACT_COUNTER Macro

File

hci.h

C

```
#define HCI_READ_FAILED_CONTACT_COUNTER HCI_OPCODE(OGF_STATUS, 0x0001)
```

Description

addtogroup hci @{

@name Status Parameters

details The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.

HCI_READ_HOLD_MODE_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_HOLD_MODE_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002B)
```

Description

This is macro HCI_READ_HOLD_MODE_ACTIVITY.

HCI_READ_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0044)
```

Description

This is macro HCI_READ_INQUIRY_MODE.

HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0058)
```

Description

This is macro HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL.

HCI_READ_INQUIRY_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001D)
```

Description

This is macro HCI_READ_INQUIRY_SCAN_ACTIVITY.

HCI_READ_INQUIRY_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0042)
```

Description

This is macro HCI_READ_INQUIRY_SCAN_TYPE.

HCI_READ_LE_HOST_SUPPORT Macro

File

hci.h

C

```
#define HCI_READ_LE_HOST_SUPPORT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006C)
```

Description

This is macro HCI_READ_LE_HOST_SUPPORT.

HCI_READ_LINK_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_READ_LINK_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000C)
```

Description

This is macro HCI_READ_LINK_POLICY_SETTINGS.

HCI_READ_LINK_QUALITY Macro

File

hci.h

C

```
#define HCI_READ_LINK_QUALITY HCI_OPCODE(OGF_STATUS, 0x0003)
```

Description

This is macro HCI_READ_LINK_QUALITY.

HCI_READ_LINK_SUPERVISION_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_LINK_SUPERVISION_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0036)
```

Description

This is macro HCI_READ_LINK_SUPERVISION_TIMEOUT.

HCI_READ_LMP_HANDLE Macro

File

hci.h

C

```
#define HCI_READ_LMP_HANDLE HCI_OPCODE(OGF_LINK_CONTROL, 0x0020)
```

Description

This is macro HCI_READ_LMP_HANDLE.

HCI_READ_LOCAL_EXTENDED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_EXTENDED_FEATURES HCI_OPCODE(OGF_INFORMATION, 0x0004)
```

Description

This is macro HCI_READ_LOCAL_EXTENDED_FEATURES.

HCI_READ_LOCAL_NAME Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_NAME HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0014)
```

Description

This is macro HCI_READ_LOCAL_NAME.

HCI_READ_LOCAL_OOB_DATA Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_OOB_DATA HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0057)
```

Description

This is macro HCI_READ_LOCAL_OOB_DATA.

HCI_READ_LOCAL_SUPPORTED_COMMANDS Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_SUPPORTED_COMMANDS HCI_OPCODE(OGF_INFORMATION, 0x0002)
```

Description

This is macro HCI_READ_LOCAL_SUPPORTED_COMMANDS.

HCI_READ_LOCAL_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_SUPPORTED_FEATURES HCI_OPCODE(OGF_INFORMATION, 0x0003)
```

Description

This is macro HCI_READ_LOCAL_SUPPORTED_FEATURES.

HCI_READ_LOCAL_VERSION_INFORMATION Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_VERSION_INFORMATION HCI_OPCODE(OGF_INFORMATION, 0x0001)
```

Description

addtogroup hci @{

@name Informational Parameters

details The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.

HCI_READ_LOOPBACK_MODE Macro

File

hci.h

C

```
#define HCI_READ_LOOPBACK_MODE HCI_OPCODE(OGF_TESTING, 0x0001)
```

Description

addtogroup hci @{

@name Testing Commands

details The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.

HCI_READ_NUM_BROADCST_RETR Macro

File

hci.h

C

```
#define HCI_READ_NUM_BROADCST_RETR HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0029)
```

Description

This is macro HCI_READ_NUM_BROADCST_RETR.

HCI_READ_NUM_OF_SUPPORTED_IAC Macro

File

hci.h

C

```
#define HCI_READ_NUM_OF_SUPPORTED_IAC HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0038)
```

Description

This is macro HCI_READ_NUM_OF_SUPPORTED_IAC.

HCI_READ_PAGE_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001B)
```

Description

This is macro HCI_READ_PAGE_SCAN_ACTIVITY.

HCI_READ_PAGE_SCAN_PERIOD_MODE Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_PERIOD_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003B)
```

Description

This is macro HCI_READ_PAGE_SCAN_PERIOD_MODE.

HCI_READ_PAGE_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0046)
```

Description

This is macro HCI_READ_PAGE_SCAN_TYPE.

HCI_READ_PAGE_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_PAGE_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0017)
```

Description

This is macro HCI_READ_PAGE_TIMEOUT.

HCI_READ_PIN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_PIN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0009)
```

Description

This is macro HCI_READ_PIN_TYPE.

HCI_READ_REFRESH_ENCRYPTION_KEY Macro

File

hci.h

C

```
#define HCI_READ_REFRESH_ENCRYPTION_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0053)
```

Description

This is macro HCI_READ_REFRESH_ENCRYPTION_KEY.

HCI_READ_REMOTE_EXTENDED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_EXTENDED_FEATURES HCI_OPCODE(OGF_LINK_CONTROL, 0x001C)
```

Description

This is macro HCI_READ_REMOTE_EXTENDED_FEATURES.

HCI_READ_REMOTE_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_SUPPORTED_FEATURES HCI_OPCODE(OGF_LINK_CONTROL, 0x001B)
```

Description

This is macro HCI_READ_REMOTE_SUPPORTED_FEATURES.

HCI_READ_REMOTE_VERSION_INFORMATION Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_VERSION_INFORMATION HCI_OPCODE(OGF_LINK_CONTROL, 0x001D)
```

Description

This is macro HCI_READ_REMOTE_VERSION_INFORMATION.

HCI_READ_RSSI Macro

File

hci.h

C

```
#define HCI_READ_RSSI HCI_OPCODE(OGF_STATUS, 0x0005)
```

Description

This is macro HCI_READ_RSSI.

HCI_READ_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_SCAN_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0019)
```

Description

This is macro HCI_READ_SCAN_ENABLE.

HCI_READ_SIMPLE_PAIRING_MODE Macro

File

hci.h

C

```
#define HCI_READ_SIMPLE_PAIRING_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0055)
```

Description

This is macro HCI_READ_SIMPLE_PAIRING_MODE.

HCI_READ_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_READ_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000D)
```

Description

This is macro HCI_READ_STORED_LINK_KEY.

HCI_READ_SYNC_FLOW_CONTROL_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_SYNC_FLOW_CONTROL_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002E)
```

Description

This is macro HCI_READ_SYNC_FLOW_CONTROL_ENABLE.

HCI_READ_TRANSMIT_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_READ_TRANSMIT_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002D)
```

Description

This is macro HCI_READ_TRANSMIT_POWER_LEVEL.

HCI_READ_VOICE_SETTING Macro

File

hci.h

C

```
#define HCI_READ_VOICE_SETTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0025)
```

Description

This is macro HCI_READ_VOICE_SETTING.

HCI_REJECT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_REJECT_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x000A)
```

Description

This is macro HCI_REJECT_CONNECTION_REQUEST.

HCI_REJECT_SYNCH_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_REJECT_SYNCH_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x002A)
```

Description

This is macro HCI_REJECT_SYNCH_CONNECTION_REQUEST.

HCI_REMOTE_NAME_REQUEST Macro

File

hci.h

C

```
#define HCI_REMOTE_NAME_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0019)
```

Description

This is macro HCI_REMOTE_NAME_REQUEST.

HCI_REMOTE_NAME_REQUEST_CANCEL Macro

File

hci.h

C

```
#define HCI_REMOTE_NAME_REQUEST_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x001A)
```

Description

This is macro HCI_REMOTE_NAME_REQUEST_CANCEL.

HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0033)
```

Description

This is macro HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY.

HCI_REMOTE_OOB_DATA_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_REMOTE_OOB_DATA_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0030)
```

Description

This is macro HCI_REMOTE_OOB_DATA_REQUEST_REPLY.

HCI_RESET Macro

File

hci.h

C

```
#define HCI_RESET HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0003)
```

Description

This is macro HCI_RESET.

HCI_RESET_FAILED_CONTACT_COUNTER Macro

File

hci.h

C

```
#define HCI_RESET_FAILED_CONTACT_COUNTER HCI_OPCODE(OGF_STATUS, 0x0002)
```

Description

This is macro HCI_RESET_FAILED_CONTACT_COUNTER.

HCI_ROLE_DISCOVERY Macro

File

hci.h

C

```
#define HCI_ROLE_DISCOVERY HCI_OPCODE(OGF_LINK_POLICY, 0x0009)
```

Description

This is macro HCI_ROLE_DISCOVERY.

HCI_ROLE_SWITCH_ALLOW Macro

File

hci.h

C

```
#define HCI_ROLE_SWITCH_ALLOW 0x01
```

Description

This is macro HCI_ROLE_SWITCH_ALLOW.

HCI_ROLE_SWITCH_DISALLOW Macro

File

hci.h

C

```
#define HCI_ROLE_SWITCH_DISALLOW 0x00
```

Description

This is macro HCI_ROLE_SWITCH_DISALLOW.

HCI_SCAN_INQUIRY Macro

File

hci.h

C

```
#define HCI_SCAN_INQUIRY 0x01
```

Description

Flags for HCI_READ_SCAN_ENABLE/HCI_WRITE_SCAN_ENABLE

HCI_SCAN_PAGE Macro

File

hci.h

C

```
#define HCI_SCAN_PAGE 0x02
```

Description

This is macro HCI_SCAN_PAGE.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW 0x2
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD 0x0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW 0x1
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT 0x3
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW 0x200
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR 0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW 0x100
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT 0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT 0x040
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE 0x080
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED 0x0C0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED.

HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16 Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16 0x020
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16.

HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8 Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8 0x0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8.

HCI_SCO_DATA_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_SCO_DATA_HEADER_LEN 3
```

Description

This is macro HCI_SCO_DATA_HEADER_LEN.

HCI_SCO_MAX_DATA_LEN Macro

File

hci.h

C

```
#define HCI_SCO_MAX_DATA_LEN 255
```

Description

This is macro HCI_SCO_MAX_DATA_LEN.

HCI_SCO_MAX_LATENCY_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_MAX_LATENCY_DONTCARE 0xFFFF
```

Description

This is macro HCI_SCO_MAX_LATENCY_DONTCARE.

HCI_SCO_PACKET_TYPE_ALL Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_ALL 0x003F
```

Description

This is macro HCI_SCO_PACKET_TYPE_ALL.

HCI_SCO_PACKET_TYPE_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV3 0x0008
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV3.

HCI_SCO_PACKET_TYPE_EV4 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV4 0x0010
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV4.

HCI_SCO_PACKET_TYPE_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV5 0x0020
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV5.

HCI_SCO_PACKET_TYPE_HV1 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV1 0x0001
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV1.

HCI_SCO_PACKET_TYPE_HV2 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV2 0x0002
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV2.

HCI_SCO_PACKET_TYPE_HV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV3 0x0004
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV3.

HCI_SCO_PACKET_TYPE_NO_2_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_2_EV3 0x0040
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_2_EV3.

HCI_SCO_PACKET_TYPE_NO_2_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_2_EV5 0x0100
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_2_EV5.

HCI_SCO_PACKET_TYPE_NO_3_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_3_EV3 0x0080
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_3_EV3.

HCI_SCO_PACKET_TYPE_NO_3_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_3_EV5 0x0200
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_3_EV5.

HCI_SCO_RTX_EFFORT_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_DONTCARE 0xFF
```

Description

This is macro HCI_SCO_RTX_EFFORT_DONTCARE.

HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION 0x0
```

Description

This is macro HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION.

HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY 0x2
```

Description

This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY.

HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION 0x1
```

Description

This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION.

HCI_SCO_RX_BANDWIDTH_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_RX_BANDWIDTH_DONTCARE 0xFFFFFFFF
```

Description

This is macro HCI_SCO_RX_BANDWIDTH_DONTCARE.

HCI_SCO_TX_BANDWIDTH_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_TX_BANDWIDTH_DONTCARE 0xFFFFFFFF
```

Description

This is macro HCI_SCO_TX_BANDWIDTH_DONTCARE.

HCI_SEND_DATA_STATUS_INTERRUPTED Macro

File

hci.h

C

```
#define HCI_SEND_DATA_STATUS_INTERRUPTED 1
```

Description

This is macro HCI_SEND_DATA_STATUS_INTERRUPTED.

HCI_SEND_DATA_STATUS_SUCCESS Macro

File

hci.h

C

```
#define HCI_SEND_DATA_STATUS_SUCCESS 0
```

Description

This is macro HCI_SEND_DATA_STATUS_SUCCESS.

HCI_SEND_KEY_PRESS_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_SEND_KEY_PRESS_NOTIFICATION HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0060)
```

Description

This is macro HCI_SEND_KEY_PRESS_NOTIFICATION.

HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION Macro

File

hci.h

C

```
#define HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003F)
```

Description

This is macro HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION.

HCI_SET_CONNECTION_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_SET_CONNECTION_ENCRYPTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0013)
```

Description

This is macro HCI_SET_CONNECTION_ENCRYPTION.

HCI_SET_CTRL_TO_HOST_FLOW_CONTROL Macro

File

hci.h

C

```
#define HCI_SET_CTRL_TO_HOST_FLOW_CONTROL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0031)
```

Description

This is macro HCI_SET_CTRL_TO_HOST_FLOW_CONTROL.

HCI_SET_EVENT_FILTER Macro

File

hci.h

C

```
#define HCI_SET_EVENT_FILTER HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0005)
```

Description

This is macro HCI_SET_EVENT_FILTER.

HCI_SET_EVENT_MASK Macro

File

hci.h

C

```
#define HCI_SET_EVENT_MASK HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0001)
```

Description

addtogroup hci @{

@name Controller & Baseband commands

details The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.

HCI_SETUP_SYNCHRONOUS_CONNECTION Macro

File

hci.h

C

```
#define HCI_SETUP_SYNCHRONOUS_CONNECTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0028)
```

Description

This is macro HCI_SETUP_SYNCHRONOUS_CONNECTION.

HCI_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_SNIFF_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0003)
```

Description

This is macro HCI_SNIFF_MODE.

HCI_SNIFF_SUBRATING Macro

File

hci.h

C

```
#define HCI_SNIFF_SUBRATING HCI_OPCODE(OGF_LINK_POLICY, 0x0011)
```

Description

This is macro HCI_SNIFF_SUBRATING.

HCI_SWITCH_ROLE Macro

File

hci.h

C

```
#define HCI_SWITCH_ROLE HCI_OPCODE(OGF_LINK_POLICY, 0x000B)
```

Description

This is macro HCI_SWITCH_ROLE.

HCI_TRANSPORT_HEADER_LEN Macro

File

hci_transport.h

C

```
#define HCI_TRANSPORT_HEADER_LEN 1
```

Description

This is macro HCI_TRANSPORT_HEADER_LEN.

HCI_UART_PACKET_TYPE_ACL_DATA Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_ACL_DATA 2
```

Description

This is macro HCI_UART_PACKET_TYPE_ACL_DATA.

HCI_UART_PACKET_TYPE_COMMAND Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_COMMAND 1
```

Description

defgroup hcitr_uart HCI UART (H4) transport protocol ingroup hcitr

details This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application: li bt_oem_send() li bt_oem_recv()

HCI_UART_PACKET_TYPE_EVENT Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_EVENT 4
```

Description

This is macro HCI_UART_PACKET_TYPE_EVENT.

HCI_UART_PACKET_TYPE_SCO_DATA Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_SCO_DATA 3
```

Description

This is macro HCI_UART_PACKET_TYPE_SCO_DATA.

HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002D)
```

Description

This is macro HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY.

HCI_USER_CONFIRMATION_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_USER_CONFIRMATION_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002C)
```

Description

This is macro HCI_USER_CONFIRMATION_REQUEST_REPLY.

HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002F)
```

Description

This is macro HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY.

HCI_USER_PASSKEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_USER_PASSKEY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002E)
```

Description

This is macro HCI_USER_PASSKEY_REQUEST_REPLY.

HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0049)
```

Description

This is macro HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE.

HCI_WRITE_AUTHENTICATION_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_AUTHENTICATION_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0020)
```

Description

This is macro HCI_WRITE_AUTHENTICATION_ENABLE.

HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0028)
```

Description

This is macro HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT.

HCI_WRITE_CLASS_OF_DEVICE Macro

File

hci.h

C

```
#define HCI_WRITE_CLASS_OF_DEVICE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0024)
```

Description

This is macro HCI_WRITE_CLASS_OF_DEVICE.

HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0016)
```

Description

This is macro HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT.

HCI_WRITE_CURRENT_IAC_LAP Macro

File

hci.h

C

```
#define HCI_WRITE_CURRENT_IAC_LAP HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003A)
```

Description

This is macro HCI_WRITE_CURRENT_IAC_LAP.

HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING Macro

File

hci.h

C

```
#define HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005B)
```

Description

This is macro HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING.

HCI_WRITE_DEFAULT_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_WRITE_DEFAULT_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000F)
```

Description

This is macro HCI_WRITE_DEFAULT_POLICY_SETTINGS.

HCI_WRITE_ENCRYPTION_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_ENCRYPTION_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0022)
```

Description

This is macro HCI_WRITE_ENCRYPTION_MODE.

HCI_WRITE_EXTENDED_INQUIRY_RESPONSE Macro

File

hci.h

C

```
#define HCI_WRITE_EXTENDED_INQUIRY_RESPONSE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0052)
```

Description

This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE.

HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN 241
```

Description

This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN.

HCI_WRITE_HOLD_MODE_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_HOLD_MODE_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002C)
```

Description

This is macro HCI_WRITE_HOLD_MODE_ACTIVITY.

HCI_WRITE_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0045)
```

Description

This is macro HCI_WRITE_INQUIRY_MODE.

HCI_WRITE_INQUIRY_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001E)
```

Description

This is macro HCI_WRITE_INQUIRY_SCAN_ACTIVITY.

HCI_WRITE_INQUIRY_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0043)
```

Description

This is macro HCI_WRITE_INQUIRY_SCAN_TYPE.

HCI_WRITE_INQUIRY_TX_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_TX_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0059)
```

Description

This is macro HCI_WRITE_INQUIRY_TX_POWER_LEVEL.

HCI_WRITE_LE_HOST_SUPPORT Macro

File

hci.h

C

```
#define HCI_WRITE_LE_HOST_SUPPORT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006D)
```

Description

This is macro HCI_WRITE_LE_HOST_SUPPORT.

HCI_WRITE_LINK_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_WRITE_LINK_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000D)
```

Description

This is macro HCI_WRITE_LINK_POLICY_SETTINGS.

HCI_WRITE_LINK_SUPERVISION_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_LINK_SUPERVISION_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0037)
```

Description

This is macro HCI_WRITE_LINK_SUPERVISION_TIMEOUT.

HCI_WRITE_LOCAL_NAME Macro

File

hci.h

C

```
#define HCI_WRITE_LOCAL_NAME HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0013)
```

Description

This is macro HCI_WRITE_LOCAL_NAME.

HCI_WRITE_LOCAL_NAME_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_WRITE_LOCAL_NAME_PARAM_LEN 248
```

Description

This is macro HCI_WRITE_LOCAL_NAME_PARAM_LEN.

HCI_WRITE_LOOPBACK_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_LOOPBACK_MODE HCI_OPCODE(OGF_TESTING, 0x0002)
```

Description

This is macro HCI_WRITE_LOOPBACK_MODE.

HCI_WRITE_NUM_BROADCST_RETR Macro

File

hci.h

C

```
#define HCI_WRITE_NUM_BROADCST_RETR HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002A)
```

Description

This is macro HCI_WRITE_NUM_BROADCST_RETR.

HCI_WRITE_PAGE_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001C)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_ACTIVITY.

HCI_WRITE_PAGE_SCAN_PERIOD_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_PERIOD_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003C)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_PERIOD_MODE.

HCI_WRITE_PAGE_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0047)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_TYPE.

HCI_WRITE_PAGE_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0018)
```

Description

This is macro HCI_WRITE_PAGE_TIMEOUT.

HCI_WRITE_PIN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_PIN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000A)
```

Description

This is macro HCI_WRITE_PIN_TYPE.

HCI_WRITE_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_SCAN_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001A)
```

Description

This is macro HCI_WRITE_SCAN_ENABLE.

HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE HCI_OPCODE(OGF_TESTING, 0x0004)
```

Description

This is macro HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE.

HCI_WRITE_SIMPLE_PAIRING_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_SIMPLE_PAIRING_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0056)
```

Description

This is macro HCI_WRITE_SIMPLE_PAIRING_MODE.

HCI_WRITE_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_WRITE_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0011)
```

Description

This is macro HCI_WRITE_STORED_LINK_KEY.

HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002F)
```

Description

This is macro HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE.

HCI_WRITE_VOICE_SETTING Macro

File

hci.h

C

```
#define HCI_WRITE_VOICE_SETTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0026)
```

Description

This is macro HCI_WRITE_VOICE_SETTING.

HCITR_3WIRE_DEFAULT_ACK_TIMEOUT Macro

File

hcitr_3wire.h

C

```
#define HCITR_3WIRE_DEFAULT_ACK_TIMEOUT 250
```

Description

This is macro HCITR_3WIRE_DEFAULT_ACK_TIMEOUT.

HCITR_BCSP_DEFAULT_ACK_TIMEOUT Macro

File

hcitr_bcsp.h

C

```
#define HCITR_BCSP_DEFAULT_ACK_TIMEOUT 250
```

Description

This is macro HCITR_BCSP_DEFAULT_ACK_TIMEOUT.

LM_PACKET_TYPE_DH1 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH1 0x0010
```

Description

This is macro LM_PACKET_TYPE_DH1.

LM_PACKET_TYPE_DH3 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH3 0x0800
```

Description

This is macro LM_PACKET_TYPE_DH3.

LM_PACKET_TYPE_DH5 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH5 0x8000
```

Description

This is macro LM_PACKET_TYPE_DH5.

LM_PACKET_TYPE_DM1 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM1 0x0008
```

Description

This is macro LM_PACKET_TYPE_DM1.

LM_PACKET_TYPE_DM3 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM3 0x0400
```

Description

This is macro LM_PACKET_TYPE_DM3.

LM_PACKET_TYPE_DM5 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM5 0x4000
```

Description

This is macro LM_PACKET_TYPE_DM5.

OGF_CTRL_BASEBAND Macro

File

hci.h

C

```
#define OGF_CTRL_BASEBAND 0x03
```

Description

This is macro OGF_CTRL_BASEBAND.

OGF_INFORMATION Macro

File

hci.h

C

```
#define OGF_INFORMATION 0x04
```

Description

This is macro OGF_INFORMATION.

OGF_LE Macro

File

hci.h

C

```
#define OGF_LE 0x08
```

Description

This is macro OGF_LE.

OGF_LINK_CONTROL Macro

File

hci.h

C

```
#define OGF_LINK_CONTROL 0x01
```

Description

This is macro OGF_LINK_CONTROL.

OGF_LINK_POLICY Macro

File

hci.h

C

```
#define OGF_LINK_POLICY 0x02
```

Description

This is macro OGF_LINK_POLICY.

OGF_STATUS Macro

File

hci.h

C

```
#define OGF_STATUS 0x05
```

Description

This is macro OGF_STATUS.

OGF_TESTING Macro

File

hci.h

C

```
#define OGF_TESTING 0x06
```

Description

This is macro OGF_TESTING.

OGF_VENDOR Macro

File

hci.h

C

```
#define OGF_VENDOR 0x3F
```

Description

This is macro OGF_VENDOR.

bt_hci_cmd_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_cmd_callback_fp)(bt_int, struct _bt_hci_command_s*, struct _bt_hci_event_s*);
```

Description

This is type bt_hci_cmd_callback_fp.

bt_hci_cmd_listener_fp Type

File

hci_ctrl_state.h

C

```
typedef void (* bt_hci_cmd_listener_fp)(bt_uint event_id, bt_hci_command_t* cmd, void* cb_param);
```

Description

This is type bt_hci_cmd_listener_fp.

bt_hci_command_p Structure

File

hci.h

C

```
typedef struct _bt_hci_command_s {
  struct _bt_hci_command_s* next_cmd;
  bt_int opcode;
  bt_int params_len;
  bt_byte* params;
  bt_hci_cmd_callback_fp callback;
  void* callback_param;
  bt_int packetNum;
} bt_hci_command_t, * bt_hci_command_p;
```

Description

This is type bt_hci_command_p.

bt_hci_command_t Structure

File

hci.h

C

```
typedef struct _bt_hci_command_s {
  struct _bt_hci_command_s* next_cmd;
  bt_int opcode;
  bt_int params_len;
  bt_byte* params;
  bt_hci_cmd_callback_fp callback;
  void* callback_param;
  bt_int packetNum;
} bt_hci_command_t, * bt_hci_command_p;
```

Description

This is type bt_hci_command_t.

bt_hci_conn_state_p Type

File

hci.h

C

```
typedef struct _bt_hci_conn_state_s * bt_hci_conn_state_p;
```

Description

This is type bt_hci_conn_state_p.

bt_hci_conn_state_t Type

File

hci.h

C

```
typedef struct _bt_hci_conn_state_s bt_hci_conn_state_t;
```

Description

This is type bt_hci_conn_state_t.

bt_hci_connect_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_connect_callback_fp)(bt_byte status, bt_hci_conn_state_t *pconn, void* param);
```

Description

brief HCI connect callback. ingroup hci

details This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete.

param status Operation status. c It is 0 if connection was successfully established. param pconn pointer to a structure representing the established connection. param param pointer to arbitrary data passed to the bt_hci_connect() function through its c param parameter.

bt_hci_ctrl_listener_t Type

File

hci_ctrl_state.h

C

```
typedef struct _bt_hci_ctrl_listener_t bt_hci_ctrl_listener_t;
```

Description

This is type bt_hci_ctrl_listener_t.

bt_hci_ctrl_state_t Structure

File

hci_ctrl_state.h

C

```
typedef struct _bt_hci_ctrl_state_s {
  bt_byte state;
  bt_byte default_link_policy;
  bt_bdaddr_t bdaddr;
  bt_int num_hci_cmd_packets;
  bt_int total_num_acl_data_packets;
  bt_int total_num_sco_data_packets;
  bt_int aclDataPacketLen;
  bt_byte scoDataPacketLen;
  bt_hci_conn_state_t* connections;
  bt_ulong _discoverable_period;
  bt_ulong _connectable_period;
  bt_byte _last_cmd_status;
  bt_hci_le_ctrl_state_t* le_ctrl_state;
  bt_ulong event_mask_l;
  bt_ulong event_mask_h;
  bt_byte incoming_connection_role;
  bt_byte inquiry_response_tx_power_level;
  struct _bt_l2cap_mgr_s* l2cap_mgr;
  pf_l2cap_receive_callback l2cap_data_receive_callback;
  bt_hci_start_callback_fp _init_cb;
  void* _init_param;
  bt_hci_connect_callback_fp _connect_cb;
  void* _connect_param;
  bt_uint _connect_acl_config;
  bt_hci_connect_callback_fp _listen_cb;
  void* _listen_param;
  bt_ulong sco_tx_bandwidth;
  bt_ulong sco_rcv_bandwidth;
  bt_uint sco_max_latency;
  bt_uint sco_content_format;
  bt_byte sco_retransmission_effort;
```

```
    bt_uint sco_packet_type;
    bt_hci_connect_callback_fp sco_listen_cb;
    void* _sco_listen_param;
    bt_hci_connect_callback_fp sco_connect_cb;
    void* _sco_connect_param;
    bt_hci_inquiry_callback_fp inquiry_cb;
    bt_hci_request_remote_name_callback_fp remote_name_cb;
    bt_byte inquiry_max_responses;
    bt_byte inquiry_length;
    pf_hci_sleep_callback sleep_cb;
    pf_hci_wakeup_callback wakeup_cb;
    bt_hci_ctrl_listener_t* listeners;
  } bt_hci_ctrl_state_t;
```

Description

This is type bt_hci_ctrl_state_t.

bt_hci_data_buffer_p Structure

File

hci_data_buffer.h

C

```
typedef struct _bt_hci_data_buffer_s {
  bt_int state;
  bt_hci_data_t packet;
} bt_hci_data_buffer_t, * bt_hci_data_buffer_p;
```

Description

This is type bt_hci_data_buffer_p.

bt_hci_data_buffer_t Structure

File

hci_data_buffer.h

C

```
typedef struct _bt_hci_data_buffer_s {
  bt_int state;
  bt_hci_data_t packet;
} bt_hci_data_buffer_t, * bt_hci_data_buffer_p;
```

Description

This is type bt_hci_data_buffer_t.

bt_hci_data_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_data_callback_fp)(bt_byte status, struct _bt_hci_data_s*);
```

Description

This is type bt_hci_data_callback_fp.

bt_hci_data_listener_fp Type

File

hci_ctrl_state.h

C

```
typedef void (* bt_hci_data_listener_fp)(bt_hci_conn_state_t* connection, const bt_byte* data, bt_int len,
```

```
bt_bool sent, void* cb_param);
```

Description

This is type bt_hci_data_listener_fp.

bt_hci_data_p Structure

File

hci.h

C

```
typedef struct _bt_hci_data_s {
  struct _bt_hci_data_s* next_data;
  bt_hci_hconn_t hconn;
  bt_uint data_len;
  bt_byte_p pdata;
  bt_uint pos;
  bt_byte link_type;
  bt_hci_data_callback_fp callback;
  void * param;
  bt_int packetNum;
} bt_hci_data_t, * bt_hci_data_p;
```

Description

This is type bt_hci_data_p.

bt_hci_data_t Structure

File

hci.h

C

```
typedef struct _bt_hci_data_s {
  struct _bt_hci_data_s* next_data;
  bt_hci_hconn_t hconn;
  bt_uint data_len;
  bt_byte_p pdata;
  bt_uint pos;
  bt_byte link_type;
  bt_hci_data_callback_fp callback;
  void * param;
  bt_int packetNum;
} bt_hci_data_t, * bt_hci_data_p;
```

Description

This is type bt_hci_data_t.

bt_hci_disconnect_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_disconnect_callback_fp)(bt_byte status, bt_byte reason, bt_hci_conn_state_t *pconn,
void* param);
```

Description

brief HCI disconnect callback. ingroup hci

details This typedef defines a type for the callback function that is called when an HCI connection has been terminated.

param status Operation status. c It is 0 if connection has been successfully terminated. param reason Reason for disconnection. param pconn pointer to a structure representing the connection. param param pointer to arbitrary data associated with an event listener.

bt_hci_event_e Union

File

hci_conn_state.h

C

```
typedef union _bt_hci_event_e {
  bt_hci_evt_disconnection_complete_t disconnect;
} bt_hci_event_e;
```

Description

This is type bt_hci_event_e.

bt_hci_event_handler_ex_fp Type

File

hci_evt_handlers.h

C

```
typedef void (* bt_hci_event_handler_ex_fp)(bt_hci_event_p, bt_uint params_len);
```

Description

This is type bt_hci_event_handler_ex_fp.

bt_hci_event_handler_fp Type

File

hci_evt_handlers.h

C

```
typedef void (* bt_hci_event_handler_fp)(bt_hci_event_p);
```

Description

This is type bt_hci_event_handler_fp.

bt_hci_event_listener_fp Type

File

hci.h

C

```
typedef void (* bt_hci_event_listener_fp)(bt_int evcode, void* evt_params, void* cb_param);
```

Description

This is type bt_hci_event_listener_fp.

bt_hci_event_p Structure

File

hci.h

C

```
typedef struct _bt_hci_event_s {
  bt_int evcode;
  bt_int params_len;
  bt_byte* params;
} bt_hci_event_t, * bt_hci_event_p;
```

Description

This is type bt_hci_event_p.

bt_hci_event_t Structure

File

hci.h

C

```
typedef struct _bt_hci_event_s {
  bt_int evcode;
  bt_int params_len;
  bt_byte* params;
} bt_hci_event_t, * bt_hci_event_p;
```

Description

This is type bt_hci_event_t.

bt_hci_evt_authentication_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_authentication_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
} bt_hci_evt_authentication_complete_t;
```

Description

This is type bt_hci_evt_authentication_complete_t.

bt_hci_evt_command_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_command_complete_t {
  bt_byte status;
  bt_int opcode;
  bt_hci_command_t* cmd;
} bt_hci_evt_command_complete_t;
```

Description

This is type bt_hci_evt_command_complete_t.

bt_hci_evt_command_status_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_command_status_t {
  bt_byte status;
  bt_int opcode;
  bt_hci_command_t* cmd;
} bt_hci_evt_command_status_t;
```

Description

This is type bt_hci_evt_command_status_t.

bt_hci_evt_connection_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_connection_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_bool incomming;
} bt_hci_evt_connection_complete_t;
```

Description

This is type bt_hci_evt_connection_complete_t.

bt_hci_evt_connection_request_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_connection_request_t {
  bt_bdaddr_t bdaddr;
  bt_long dev_class;
  bt_byte link_type;
} bt_hci_evt_connection_request_t;
```

Description

This is type bt_hci_evt_connection_request_t.

bt_hci_evt_disconnection_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_disconnection_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_byte reason;
} bt_hci_evt_disconnection_complete_t;
```

Description

This is type bt_hci_evt_disconnection_complete_t.

bt_hci_evt_encryption_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_encryption_change_s {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_byte enabled;
} bt_hci_evt_encryption_change_t;
```

Description

This is type bt_hci_evt_encryption_change_t.

bt_hci_evt_mode_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_mode_change_t {
  bt_hci_conn_state_t* pconn;
  bt_byte mode;
  bt_int interval;
} bt_hci_evt_mode_change_t;
```

Description

This is type bt_hci_evt_mode_change_t.

bt_hci_evt_role_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_role_change_t {
  bt_hci_conn_state_t* pconn;
  bt_byte role;
} bt_hci_evt_role_change_t;
```

Description

This is type bt_hci_evt_role_change_t.

bt_hci_hconn_p Type

File

hci.h

C

```
typedef bt_int * bt_hci_hconn_p;
```

Description

This is type bt_hci_hconn_p.

bt_hci_hconn_t Type

File

hci.h

C

```
typedef bt_int bt_hci_hconn_t;
```

Description

This is type bt_hci_hconn_t.

bt_hci_inquiry_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_inquiry_callback_fp)(bt_byte status, bt_hci_inquiry_response_t* response);
```

Description

This is type bt_hci_inquiry_callback_fp.

bt_hci_inquiry_response_t Structure

File

hci.h

C

```
typedef struct _bt_hci_inquiry_response_t {
  bt_bdaddr_t bdaddr;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
  bt_byte* eir;
  bt_byte eir_len;
} bt_hci_inquiry_response_t;
```

Description

This is type bt_hci_inquiry_response_t.

bt_hci_le_advertising_report_t Type

File

hci_le.h

C

```
typedef struct _bt_hci_le_advertising_report_t bt_hci_le_advertising_report_t;
```

Description

This is type bt_hci_le_advertising_report_t.

bt_hci_le_conn_state_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_conn_state_t {
  bt_byte own_address_type;
  bt_byte peer_address_type;
  bt_uint conn_interval;
  bt_uint conn_latency;
  bt_uint supervision_timeout;
  bt_byte master_clock_accuracy;
} bt_hci_le_conn_state_t;
```

Description

This is type bt_hci_le_conn_state_t.

bt_hci_le_connect_parameters_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_connect_parameters_t {
  bt_uint scan_interval;
  bt_uint scan_window;
  bt_byte filter_policy;
```

```
  bt_byte own_address_type;
  bt_uint conn_interval_min;
  bt_uint conn_interval_max;
  bt_uint conn_latency;
  bt_uint supervision_timeout;
  bt_uint min_ce_length;
  bt_uint max_ce_length;
} bt_hci_le_connect_parameters_t;
```

Description

This is type bt_hci_le_connect_parameters_t.

bt_hci_le_ctrl_state_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_ctrl_state_t {
  bt_byte flags;
  bt_byte total_num_acl_data_packets;
  bt_int acl_data_packet_len;
  struct _bt_security_mgr_t* sm;
  bt_bdaddr_t bdaddr;
  bt_byte advertising_address_type;
  bt_byte connecting_address_type;
  bt_hci_le_connect_parameters_t def_connect_parameters;
  bt_le_evt_handler handler;
  bt_hci_le_scan_callback_fp scan_cb;
  void* scan_cb_param;
  bt_bool (* connect_ex)(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte
peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void *param);
  bt_bool (* set_connect_parameters)(bt_hci_le_connect_parameters_t* params);
  bt_bool (* get_connect_parameters)(bt_hci_le_connect_parameters_t* params);
  void (* sm_long_term_key_request_handler)(bt_hci_event_t* evt);
  bt_uint (* sm_get_session_state)(bt_hci_conn_state_t* conn);
  bt_byte (* sm_get_session_key_size)(bt_hci_conn_state_t* conn);
  bt_bool (* sm_authenticate)(bt_hci_conn_state_t* conn, bt_byte auth_req);
} bt_hci_le_ctrl_state_t;
```

Members

Members	Description
bt_hci_le_connect_parameters_t def_connect_parameters;	default connect parameters
bt_bool (* connect_ex)(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void *param);	API calls

Description

This is type bt_hci_le_ctrl_state_t.

bt_hci_le_evt_connection_updated_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_connection_updated_t {
  bt_byte status;
```

```
  struct _bt_hci_conn_state_s* conn;
} bt_hci_le_evt_connection_updated_t;
```

Description

This is type bt_hci_le_evt_connection_updated_t.

bt_hci_le_evt_read_remote_used_features_completed_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_read_remote_used_features_completed_t {
  bt_byte status;
  struct _bt_hci_conn_state_s* conn;
  bt_byte features[8];
} bt_hci_le_evt_read_remote_used_features_completed_t;
```

Description

This is type bt_hci_le_evt_read_remote_used_features_completed_t.

bt_hci_le_evt_read_support_params_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_read_support_params_t {
  bt_byte supported;
  bt_byte simultaneous_supported;
} bt_hci_le_evt_read_support_params_t;
```

Description

This is type bt_hci_le_evt_read_support_params_t.

bt_hci_le_scan_callback_fp Type

File

hci_le.h

C

```
typedef void (* bt_hci_le_scan_callback_fp)(bt_byte evt, bt_hci_le_advertising_report_t* report, void*
param);
```

Description

This is type bt_hci_le_scan_callback_fp.

bt_hci_link_key_t Structure

File

hci_linkkey_buffer.h

C

```
typedef struct _bt_hci_link_key_s {
  bt_bdaddr_t bdaddr;
  bt_byte linkkey[HCI_LINK_KEY_LEN];
} bt_hci_link_key_t;
```

Description

This is type bt_hci_link_key_t.

bt_hci_listener_t Type

File

hci_conn_state.h

C

```
typedef struct _bt_hci_listener_t bt_hci_listener_t;
```

Description

This is type bt_hci_listener_t.

bt_hci_read_inquiry_mode_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_mode_callback_fp)(bt_byte status, bt_byte mode);
```

Description

This is type bt_hci_read_inquiry_mode_callback_fp.

bt_hci_read_inquiry_scan_activity_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_scan_activity_callback_fp)(bt_byte status, bt_uint interval, bt_uint
window);
```

Description

This is type bt_hci_read_inquiry_scan_activity_callback_fp.

bt_hci_read_inquiry_scan_type_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_scan_type_callback_fp)(bt_byte status, bt_byte scanType);
```

Description

This is type bt_hci_read_inquiry_scan_type_callback_fp.

bt_hci_read_page_scan_activity_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_activity_callback_fp)(bt_byte status, bt_uint interval, bt_uint
window);
```

Description

This is type bt_hci_read_page_scan_activity_callback_fp.

bt_hci_read_page_scan_period_mode_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_period_mode_callback_fp)(bt_byte status, bt_byte mode);
```

Description

This is type bt_hci_read_page_scan_period_mode_callback_fp.

bt_hci_read_page_scan_type_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_type_callback_fp)(bt_byte status, bt_byte scanType);
```

Description

This is type bt_hci_read_page_scan_type_callback_fp.

bt_hci_read_page_timeout_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_timeout_callback_fp)(bt_byte status, bt_uint timeout);
```

Description

This is type bt_hci_read_page_timeout_callback_fp.

bt_hci_request_remote_name_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_request_remote_name_callback_fp)(bt_bdaddr_p pdbaddr, const char* name);
```

Description

This is type bt_hci_request_remote_name_callback_fp.

bt_hci_sco_read_data_callback_fp Type

File

hci_conn_state.h

C

```
typedef void (* bt_hci_sco_read_data_callback_fp)(bt_hci_conn_state_t* pconn, bt_byte_p data, bt_byte len);
```

Description

This is type bt_hci_sco_read_data_callback_fp.

bt_hci_start_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_start_callback_fp)(bt_bool success, void* param);
```

Description

brief HCI initialization callback. ingroup hci

details This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function.

param success Specifies whether HCI initialization succeeded or not.

bt_hci_stop_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_stop_callback_fp)(void* param);
```

Description

brief HCI stop callback. ingroup hci

details This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.

bt_hci_transport_recv_packet_callback_fp Type

File

hci_transport.h

C

```
typedef void (* bt_hci_transport_recv_packet_callback_fp)(bt_uint len);
```

Description

This is type bt_hci_transport_recv_packet_callback_fp.

bt_hci_transport_send_packet_callback_fp Type

File

hci_transport.h

C

```
typedef void (* bt_hci_transport_send_packet_callback_fp)(void);
```

Description

This is type bt_hci_transport_send_packet_callback_fp.

bt_hcitr_tih4_power_callback_fp Type

File

hcitr_tih4.h

C

```
typedef void (* bt_hcitr_tih4_power_callback_fp)(bt_hcitr_tih4_power_event_e event);
```

Description

This is type bt_hcitr_tih4_power_callback_fp.

bt_hcitr_tih4_power_event_e Enumeration

File

hcitr_tih4.h

C

```
typedef enum _hcitr_tih4_power_event_e {
  HCITR_TIH4_POWER_EVENT_PREPARE_TO_SLEEP,
  HCITR_TIH4_POWER_EVENT_SLEEP,
  HCITR_TIH4_POWER_EVENT_WAKE_UP,
  HCITR_TIH4_POWER_EVENT_AWAKE
} bt_hcitr_tih4_power_event_e;
```

Description

This is type bt_hcitr_tih4_power_event_e.

bt_le_evt_handler Type

File

hci_le.h

C

```
typedef void (* bt_le_evt_handler)(bt_hci_event_t* evt);
```

Description

This is type bt_le_evt_handler.

hci_transport_t Structure

File

hci_transport.h

C

```
typedef struct _hci_transport_t {
  void (* send_packet)(const bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
  void (* recv_packet)(bt_byte* buffer, bt_uint len, bt_hci_transport_recv_packet_callback_fp callback);
} hci_transport_t;
```

Description

This is type hci_transport_t.

pf_hci_sleep_callback Type

File

hci.h

C

```
typedef void (* pf_hci_sleep_callback)(bt_bool status);
```

Description

This is type pf_hci_sleep_callback.

pf_hci_wakeup_callback Type

File

hci.h

C

```
typedef void (* pf_hci_wakeup_callback)(bt_bool status);
```

Description

This is type pf_hci_wakeup_callback.

pf_l2cap_receive_callback Type

File

hci_ctrl_state.h

C

```
typedef void (* pf_l2cap_receive_callback)(struct _bt_l2cap_mgr_s*, bt_hci_conn_state_p, bt_byte_p, bt_int
len);
```

Description

This is type pf_l2cap_receive_callback.

_bt_hci_conn_state_s Structure

File

hci_conn_state.h

C

```
struct _bt_hci_conn_state_s {
  bt_hci_hconn_t hconn;
  bt_bdaddr_t bdaddr_remote;
  bt_byte type;
  bt_byte state;
  bt_byte mode;
  bt_byte role;
  bt_long l2cap_channel_close_time;
  bt_uint acl_config;
  bt_byte link_type;
  bt_signal_t signal;
  bt_int outstanding_acl_packet_count;
  bt_hci_le_conn_state_t* le_conn_state;
  bt_ulong data_rx_time;
  bt_ulong data_tx_time;
  bt_byte idle;
  bt_queue_element_t* queue;
  bt_byte* recv_data;
  bt_int recv_data_len;
  bt_int recv_data_pos;
  bt_hci_sco_read_data_callback_fp sco_read_data_callback;
  void* sco_read_data_param;
  bt_hci_listener_t* listeners;
};
```

Members

Members	Description
bt_queue_element_t* queue;	queue for sending outgoing data packets
bt_byte* recv_data;	buffer to receive incoming packets
bt_hci_sco_read_data_callback_fp sco_read_data_callback;	bt_hci_disconnect_callback_fp hci_disconnect_callback; void *hci_disconnect_param;

Description

This is record _bt_hci_conn_state_s.

_bt_hci_ctrl_listener_t Structure

File

hci_ctrl_state.h

C

```
struct _bt_hci_ctrl_listener_t {
  bt_hci_ctrl_listener_t* next_listener;
  bt_byte listener_type;
  bt_uint event_id;
  union {
    void* ptr;
    bt_hci_event_listener_fp hci_event;
    bt_hci_cmd_listener_fp cmd_event;
    bt_hci_data_listener_fp data;
  } callback;
  void* callback_param;
};
```

Description

This is record _bt_hci_ctrl_listener_t.

_bt_hci_le_advertising_report_t Structure

File

hci_le.h

C

```
struct _bt_hci_le_advertising_report_t {
  bt_byte event_type;
  bt_byte address_type;
  bt_bdaddr_t bdaddr;
  bt_byte data_len;
  bt_byte* data;
  bt_byte rssi;
};
```

Description

This is record _bt_hci_le_advertising_report_t.

_bt_hci_listener_t Structure

File

hci_conn_state.h

C

```
struct _bt_hci_listener_t {
  bt_hci_listener_t* next_listener;
  bt_byte event_id;
  union {
    void* ptr;
    bt_hci_disconnect_callback_fp disconnect;
  } callback;
  void* callback_param;
};
```

Description

This is record _bt_hci_listener_t.

HCI_DISCOVERABLE_MODE_GENERAL Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE_MODE_GENERAL 0
```

Description

This is macro HCI_DISCOVERABLE_MODE_GENERAL.

HCI_DISCOVERABLE_MODE_LIMITED Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE_MODE_LIMITED 1
```

Description

This is macro HCI_DISCOVERABLE_MODE_LIMITED.

HCI_EIR_TYPE_3D_Information_Data Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_3D_Information_Data 0x3D
```

Description

This is macro HCI_EIR_TYPE_3D_Information_Data.

HCI_EIR_TYPE_ADVERTISING_INTERVAL Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_ADVERTISING_INTERVAL 0x1A
```

Description

This is macro HCI_EIR_TYPE_ADVERTISING_INTERVAL.

HCI_EIR_TYPE_APPEARANCE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_APPEARANCE 0x19
```

Description

This is macro HCI_EIR_TYPE_APPEARANCE.

HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS 0x1B
```

Description

This is macro HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS.

HCI_EIR_TYPE_LE_ROLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_ROLE 0x1C
```

Description

This is macro HCI_EIR_TYPE_LE_ROLE.

HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE 0x22
```

Description

This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE.

HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE 0x23
```

Description

This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE.

HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS 0x17
```

Description

This is macro HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS.

HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS 0x18
```

Description

This is macro HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS.

HCI_EIR_TYPE_SERVICE_DATA Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA 0x16
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA.

HCI_EIR_TYPE_SERVICE_DATA_UUID128 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID128 0x21
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID128.

HCI_EIR_TYPE_SERVICE_DATA_UUID16 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID16 0x16
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID16.

HCI_EIR_TYPE_SERVICE_DATA_UUID32 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID32 0x20
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID32.

HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256 0x1D
```

Description

This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256.

HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256 0x1E
```

Description

This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256.

HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE 0x12
```

Description

This is macro HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE.

HCI_EIR_TYPE_SM_OOB_FLAGS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SM_OOB_FLAGS 0x11
```

Description

This is macro HCI_EIR_TYPE_SM_OOB_FLAGS.

HCI_EIR_TYPE_SM_TK_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SM_TK_VALUE 0x10
```

Description

This is macro HCI_EIR_TYPE_SM_TK_VALUE.

HCI_EIR_TYPE_SOLICITATION_UUID128_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID128_LIST 0x15
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID128_LIST.

HCI_EIR_TYPE_SOLICITATION_UUID16_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID16_LIST 0x14
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID16_LIST.

HCI_EIR_TYPE_SOLICITATION_UUID32_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID32_LIST 0x1F
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID32_LIST.

HCI_ERR_CONNECTION_TIMEOUT Macro

File

hci_errors.h

C

```
#define HCI_ERR_CONNECTION_TIMEOUT 0x08
```

Description

This is macro HCI_ERR_CONNECTION_TIMEOUT.

HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS Macro

File

hci_private.h

C

```
#define HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS 2 // The total number of ACL data packets read from the controller will be ignored.
```

Description

The total number of ACL data packets read from the controller will be ignored. The stack will assume that he controller can accept only 1 ACL packet and sends HCI_EVT_NUM_OF_COMPLETED_PACKETS for each packet it has processed. This seems to be for controller working over SDIO (at least for Marvell's 88W8777).

HCI_INIT_FLAG_SEND_HCI_RESET Macro

File

hci_private.h

C

```
#define HCI_INIT_FLAG_SEND_HCI_RESET 1 // HCI reset will be sent when bt_hci_init is called
```

Description

HCI reset will be sent when bt_hci_init is called

HCI_LE_ADV_CHANNEL_MAP_ENABLE_37 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_37 0x01
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_37.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_38 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_38 0x02
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_38.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_39 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_39 0x04
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_39.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL 0x07
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL.

HCI_LE_ADV_CHANNEL_MAP_RESERVED Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_RESERVED 0x00
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_RESERVED.

HCI_LE_ADV_TYPE_CONN_UNDIRECT Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_CONN_UNDIRECT 0x00 // Connectable undirected advertising (ADV_IND) (default)
```

Description

Connectable undirected advertising (ADV_IND) (default)

HCI_LE_ADV_TYPE_DIRECT_HIGH Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_DIRECT_HIGH 0x01 // Connectable high duty cycle directed advertising
(ADV_DIRECT_IND, high duty cycle)
```

Description

Connectable high duty cycle directed advertising (ADV_DIRECT_IND, high duty cycle)

HCI_LE_ADV_TYPE_DIRECT_LOW Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_DIRECT_LOW 0x04 // Connectable low duty cycle directed advertising (ADV_DIRECT_IND,
low duty cycle)
```

Description

Connectable low duty cycle directed advertising (ADV_DIRECT_IND, low duty cycle)

HCI_LE_ADV_TYPE_NONCONN Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_NONCONN 0x03 // Non connectable undirected advertising (ADV_NONCONN_IND)
```

Description

Non connectable undirected advertising (ADV_NONCONN_IND)

HCI_LE_ADV_TYPE_SCAN Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_SCAN 0x02 // Scannable undirected advertising (ADV_SCAN_IND)
```

Description

Scannable undirected advertising (ADV_SCAN_IND)

HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE 2
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE.

HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE 3
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE.

HCI_LE_RANDOM_ADDRESS_TYPE_STATIC Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_STATIC 1
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_STATIC.

HCI_LINK_POLICY_ENABLE_ALL Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_ALL 0x000F
```

Description

This is macro HCI_LINK_POLICY_ENABLE_ALL.

bt_hci_le_cancel_connect Macro

File

hci_le.h

C

```
#define bt_hci_le_cancel_connect(cb) bt_hci_le_cancel_connect_ex(cb, NULL)
```

Description

This is macro bt_hci_le_cancel_connect.

bt_hci_send_pin_code Macro

File

hci.h

C

```
#define bt_hci_send_pin_code(pbdaddr_remote, pincode) bt_hci_send_pin_code_ex(pbdaddr_remote, pincode,
NULL, NULL)
```

Description

This is macro bt_hci_send_pin_code.

HCI_ERR_PAIRING_NOT_ALLOWED Macro

File

hci_errors.h

C

```
#define HCI_ERR_PAIRING_NOT_ALLOWED 0x18
```

Description

This is macro HCI_ERR_PAIRING_NOT_ALLOWED.

HCI_ERR_PIN_OR_KEY_MISSING Macro

File

hci_errors.h

C

```
#define HCI_ERR_PIN_OR_KEY_MISSING 0x06
```

Description

This is macro HCI_ERR_PIN_OR_KEY_MISSING.

HCI_PACKET_BOUNDARY_COMPLETE Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_COMPLETE 0x3
```

Description

This is macro HCI_PACKET_BOUNDARY_COMPLETE.

HCI_SUCCESS Macro

File

hci_errors.h

C

```
#define HCI_SUCCESS 0x00
```

Description

This is macro HCI_SUCCESS.

L2CAP Functions

bt_l2cap_alloc_cmd_buffer Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_header_p bt_l2cap_alloc_cmd_buffer(bt_byte code);
```

Description

This is function bt_l2cap_alloc_cmd_buffer.

bt_l2cap_alloc_cmd_config_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_config_req_p bt_l2cap_alloc_cmd_config_req();
```

Description

This is function bt_l2cap_alloc_cmd_config_req.

bt_l2cap_alloc_cmd_config_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_config_res_p bt_l2cap_alloc_cmd_config_res();
```

Description

This is function bt_l2cap_alloc_cmd_config_res.

bt_l2cap_alloc_cmd_conn_param_update_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_conn_param_update_req_t* bt_l2cap_alloc_cmd_conn_param_update_req();
```

Description

This is function bt_l2cap_alloc_cmd_conn_param_update_req.

bt_l2cap_alloc_cmd_conn_param_update_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_conn_param_update_res_t* bt_l2cap_alloc_cmd_conn_param_update_res();
```

Description

This is function bt_l2cap_alloc_cmd_conn_param_update_res.

bt_l2cap_alloc_cmd_connection_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_connection_req_p bt_l2cap_alloc_cmd_connection_req();
```

Description

This is function bt_l2cap_alloc_cmd_connection_req.

bt_l2cap_alloc_cmd_connection_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_connection_res_p bt_l2cap_alloc_cmd_connection_res();
```

Description

This is function bt_l2cap_alloc_cmd_connection_res.

bt_l2cap_alloc_cmd_disconnection_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_disconnection_req_p bt_l2cap_alloc_cmd_disconnection_req();
```

Description

This is function bt_l2cap_alloc_cmd_disconnection_req.

bt_l2cap_alloc_cmd_disconnection_res Function

File

cmdbuffer.h

C

```
pcmd_disconnection_res bt_l2cap_alloc_cmd_disconnection_res();
```

Description

This is function bt_l2cap_alloc_cmd_disconnection_res.

bt_l2cap_alloc_cmd_echo_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_echo_req_p bt_l2cap_alloc_cmd_echo_req();
```

Description

This is function bt_l2cap_alloc_cmd_echo_req.

bt_l2cap_alloc_cmd_echo_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_echo_res_p bt_l2cap_alloc_cmd_echo_res();
```

Description

This is function bt_l2cap_alloc_cmd_echo_res.

bt_l2cap_alloc_cmd_info_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_info_req_p bt_l2cap_alloc_cmd_info_req();
```

Description

This is function bt_l2cap_alloc_cmd_info_req.

bt_l2cap_alloc_cmd_info_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_info_res_p bt_l2cap_alloc_cmd_info_res();
```

Description

This is function bt_l2cap_alloc_cmd_info_res.

bt_l2cap_alloc_cmd_reject Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_reject_p bt_l2cap_alloc_cmd_reject();
```

Description

This is function bt_l2cap_alloc_cmd_reject.

bt_l2cap_alloc_frame_buffer Function

File

frame_buffer.h

C

```
bt_byte_p bt_l2cap_alloc_frame_buffer();
```

Description

This is function bt_l2cap_alloc_frame_buffer.

bt_l2cap_allocate_channel Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_allocate_channel(bt_l2cap_mgr_p pmgr, bt_int mode, bt_int type);
```

Description

This is function bt_l2cap_allocate_channel.

bt_l2cap_allocate_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
bt_l2cap_fixed_channel_t* bt_l2cap_allocate_fixed_channel(bt_l2cap_mgr_p pmgr, bt_id cid, bt_byte
link_type);
```

Description

This is function bt_l2cap_allocate_fixed_channel.

bt_l2cap_allocate_mgr Function

File

l2cap.h

C

```
bt_l2cap_mgr_t* bt_l2cap_allocate_mgr(bt_hci_ctrl_state_t* hci_ctrl);
```

Description

brief Allocate L2CAP manager. ingroup l2cap

details This function allocates and initializes an L2CAP manager structure. One L2CAP manager manages all L2CAP connections for a particular local device. The local device is specified by the c hci_ctrl parameter.

param hci_ctrl Pointer to the hci_ctrl_state structure that represents the local device (HCI controller) for which a L2CAP manager is to be allocated.

return A pointer to the allocated L2CAP manager structure. The returned L2CAP manager should be freed by a call to bt_l2cap_free_mgr when it is no longer needed.

bt_l2cap_allocate_psm Function

File

l2cap_psm.h

C

```
bt_l2cap_psm_t* bt_l2cap_allocate_psm(bt_l2cap_mgr_p pmgr, bt_int psmId);
```

Description

This is function bt_l2cap_allocate_psm.

bt_l2cap_connect_ext Function

File

l2cap.h

C

```
bt_bool bt_l2cap_connect_ext(bt_l2cap_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_int psm, bt_byte chmode,
bt_uint acl_config, bt_l2cap_connect_callback_fp connect_cb, void* param,
```

```
bt_l2cap_state_changed_callback_fp state_cb);
```

Description

brief Connect to a remote device. ingroup l2cap

details This function establishes an L2CAP connection with a remote device on a specific PSM. When connect operation completes a callback function is called.

param mgr The L2CAP manager. param remote_addr The address of the remote device. param psm The PSM for the connection. param connect_cb The Callback function that is called when the connect operation completes. param param A pointer to arbitrary data to be passed to the c connect_cb callback. param state_cb The callback function that is called when the state of the established connection changes.

return li c TRUE when the function succeeds. li c FALSE otherwise. None of the callback functions is called in this case.

bt_l2cap_connect_fixed_channel Function

File

l2cap.h

C

```
bt_bool bt_l2cap_connect_fixed_channel(bt_l2cap_mgr_p pmgr, bt_bdaddr_p pbdaddr_remote, bt_id cid, bt_uint
acl_config, bt_l2cap_connect_callback_fp cb, void* param, bt_l2cap_state_changed_callback_fp state_cb);
```

Description

This is function bt_l2cap_connect_fixed_channel.

bt_l2cap_disconnect Function

File

l2cap.h

C

```
bt_bool bt_l2cap_disconnect(bt_l2cap_channel_t* ch);
```

Description

brief Close L2CAP channel. ingroup l2cap

details This function closes an L2CAP channel. The channel can be established either by a call to bt_l2cap_connect() or by an incoming connection request.

param ch The L2CAP channel to be closed.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_echo Function

File

l2cap.h

C

```
bt_bool bt_l2cap_echo(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_byte_p data, bt_int len);
```

Description

brief Send echo command

bt_l2cap_find_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
bt_l2cap_fixed_channel_t* bt_l2cap_find_fixed_channel(const bt_l2cap_mgr_p pmgr, bt_id cid, bt_byte
link_type);
```

Description

This is function bt_l2cap_find_fixed_channel.

bt_l2cap_find_psm Function

File

l2cap_psm.h

C

```
bt_l2cap_psm_t* bt_l2cap_find_psm(const bt_l2cap_mgr_p pmgr, bt_int psmId);
```

Description

This is function bt_l2cap_find_psm.

bt_l2cap_free_cmd_buffer Function

File

cmdbuffer.h

C

```
void bt_l2cap_free_cmd_buffer(void* p);
```

Description

This is function bt_l2cap_free_cmd_buffer.

bt_l2cap_free_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
void bt_l2cap_free_fixed_channel(bt_l2cap_fixed_channel_t* fc);
```

Description

This is function bt_l2cap_free_fixed_channel.

bt_l2cap_free_frame_buffer Function

File

frame_buffer.h

C

```
void bt_l2cap_free_frame_buffer(bt_byte_p data);
```

Description

This is function bt_l2cap_free_frame_buffer.

bt_l2cap_free_mgr Function

File

l2cap.h

C

```
void bt_l2cap_free_mgr(bt_l2cap_mgr_t* mgr);
```

Description

brief Release L2CAP manger. ingroup l2cap

details This function releases the L2CAP manager structure.

param mgr The L2CAP manager structure to be released.

bt_l2cap_free_psm Function

File

l2cap_psm.h

C

```
void bt_l2cap_free_psm(bt_l2cap_psm_t* psm);
```

Description

This is function bt_l2cap_free_psm.

bt_l2cap_get_channel Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel(const bt_l2cap_mgr_p pmgr, bt_id cid);
```

Description

This is function bt_l2cap_get_channel.

bt_l2cap_get_channel_by_bdaddr_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_bdaddr_cid(const bt_l2cap_mgr_p pmgr, bt_bdaddr_t* bdaddr,
bt_id cid);
```

Description

This is function bt_l2cap_get_channel_by_bdaddr_cid.

bt_l2cap_get_channel_by_hconn_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_hconn_cid(const bt_l2cap_mgr_p pmgr, bt_hci_hconn_t hconn,
bt_id cid_dest);
```

Description

This is function bt_l2cap_get_channel_by_hconn_cid.

bt_l2cap_get_channel_by_hconn_dest_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_hconn_dest_cid(const bt_l2cap_mgr_p pmgr, bt_hci_hconn_t hconn,
bt_id cid_dest);
```

Description

This is function bt_l2cap_get_channel_by_hconn_dest_cid.

bt_l2cap_get_channel_by_psm Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_psm(const bt_l2cap_mgr_p pmgr, bt_int psm);
```

Description

This is function bt_l2cap_get_channel_by_psm.

bt_l2cap_get_mgr Function

File

l2cap.h

C

```
bt_l2cap_mgr_p bt_l2cap_get_mgr(bt_hci_hconn_t h);
```

Description

This is function bt_l2cap_get_mgr.

bt_l2cap_init Function

File

l2cap.h

C

```
void bt_l2cap_init();
```

Description

brief Initialize the L2CAP layer. ingroup l2cap

details This function initializes the L2CAP layer of the stack. It must be called prior to any other L2CAP function can be called.

bt_l2cap_init_channels Function

File

chmanager.h

C

```
void bt_l2cap_init_channels(bt_l2cap_mgr_p pmgr);
```

Description

This is function bt_l2cap_init_channels.

bt_l2cap_init_cmd_buffers Function

File

cmdbuffer.h

C

```
void bt_l2cap_init_cmd_buffers();
```

Description

This is function bt_l2cap_init_cmd_buffers.

bt_l2cap_init_frame_buffers Function

File

frame_buffer.h

C

```
void bt_l2cap_init_frame_buffers();
```

Description

This is function bt_l2cap_init_frame_buffers.

bt_l2cap_init_psms Function

File

l2cap_psm.h

C

```
void bt_l2cap_init_psms(bt_l2cap_mgr_p pmgr);
```

Description

This is function bt_l2cap_init_psms.

bt_l2cap_listen_ext Function

File

l2cap.h

C

```
bt_bool bt_l2cap_listen_ext(bt_l2cap_mgr_t* mgr, bt_int psm, bt_byte chmode, bt_uint acl_config,
bt_l2cap_listen_callback_fp callback, void* param);
```

Description

brief Listen for incoming connections. ingroup l2cap

details This function tells the L2CAP manager to listen for incoming connections on a specific PSM. When a connection is established a callback function is called.

param mgr The L2CAP manager. param psm The PSM on which the manager will listen and accept incoming connections. param callback The callback function that will be called when an incoming connection is established. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_listen_fixed_channel Function

File

l2cap.h

C

```
bt_bool bt_l2cap_listen_fixed_channel(bt_l2cap_mgr_t* mgr, bt_id cid, bt_byte link_type,
bt_l2cap_listen_callback_fp callback, void* param);
```

Description

This is function bt_l2cap_listen_fixed_channel.

bt_l2cap_packet_cmd_assembler Function

File

l2cap_packet.h

C

```
bt_int bt_l2cap_packet_cmd_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_l2cap_packet_cmd_assembler.

bt_l2cap_packet_data_assembler Function

File

l2cap_packet.h

C

```
bt_int bt_l2cap_packet_data_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_l2cap_packet_data_assembler.

bt_l2cap_read_data Function

File

channel.h

C

```
bt_bool bt_l2cap_read_data(bt_l2cap_channel_t* channel, bt_l2cap_read_data_callback_fp callback);
```

Description

This is function bt_l2cap_read_data.

bt_l2cap_reject Function

File

l2cap.h

C

```
bt_bool bt_l2cap_reject(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_byte cmd_id, bt_byte reason,
bt_l2cap_cmd_reject_param_t* param);
```

Description

brief Send reject command (used to reject unknown or invalid commands)

bt_l2cap_send_cmd Function

File

channel.h

C

```
bt_bool bt_l2cap_send_cmd(struct _bt_l2cap_mgr_s * mgr, struct _bt_hci_conn_state_s * conn,
bt_l2cap_cmd_header_p cmd, pf_l2cap_cmd_callback cb);
```

Description

This is function bt_l2cap_send_cmd.

bt_l2cap_send_data Function

File

channel.h

C

```
bt_bool bt_l2cap_send_data(bt_l2cap_channel_t* channel, const bt_byte_p data, bt_int len,
bt_l2cap_send_data_callback_fp callback, void* cb_param);
```

Description

brief Send data over an L2CAP channel ingroup l2cap

details This function sends data over the specified L2CAP channel.

param channel The L2CAP channel to send data over. param data The pointer to the data. param len The length of the data. param callback The callback function that is called when sending the data has been completed.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_update_conn_parameters Function

File

l2cap.h

C

```
bt_bool bt_l2cap_update_conn_parameters(bt_l2cap_channel_t* channel, bt_uint min_interval, bt_uint
max_interval, bt_uint slave_latency, bt_uint supervision_timeout);
```

Description

This is function bt_l2cap_update_conn_parameters.

_bt_l2cap_clear_channel_cmd_queue Function

File

l2cap_private.h

C

```
void _bt_l2cap_clear_channel_cmd_queue(bt_l2cap_channel_t* channel);
```

Description

This is function _bt_l2cap_clear_channel_cmd_queue.

_bt_l2cap_eretr_handle_xmit_event Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_handle_xmit_event(bt_l2cap_xmit_event_param_t* param);
```

Description

This is function _bt_l2cap_eretr_handle_xmit_event.

_bt_l2cap_eretr_retr_frames Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_retr_frames(bt_l2cap_channel_t* pch, bt_byte fbit);
```

Description

This is function _bt_l2cap_eretr_retr_frames.

_bt_l2cap_fcs Function

File

l2cap_eretr.h

C

```
bt_uint _bt_l2cap_fcs(const bt_byte* message, bt_int len, bt_uint start_crc);
```

Description

This is function _bt_l2cap_fcs.

_bt_l2cap_get_tick_count Function

File

l2cap_timer.h

C

```
bt_long _bt_l2cap_get_tick_count();
```

Description

This is function _bt_l2cap_get_tick_count.

_bt_l2cap_init_signal Function

File

l2cap_signal.h

C

```
void _bt_l2cap_init_signal();
```

Description

This is function _bt_l2cap_init_signal.

_bt_l2cap_init_timer Function

File

l2cap_timer.h

C

```
void _bt_l2cap_init_timer();
```

Description

This is function _bt_l2cap_init_timer.

_bt_l2cap_notify_and_remove Function

File

l2cap_command_queue.h

C

```
void _bt_l2cap_notify_and_remove(struct _bt_l2cap_mgr_s* pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _bt_l2cap_notify_and_remove.

_bt_l2cap_send_commands_from_queue Function

File

l2cap_command_queue.h

C

```
void _bt_l2cap_send_commands_from_queue(struct _bt_l2cap_mgr_s* pmgr);
```

Description

This is function _bt_l2cap_send_commands_from_queue.

_bt_l2cap_set_signal Function

File

l2cap_signal.h

C

```
void _bt_l2cap_set_signal();
```

Description

This is function _bt_l2cap_set_signal.

_l2cap_allocate_buffers Function

File

l2cap_private.h

C

```
void _l2cap_allocate_buffers();
```

Description

Defined by OEM through library configuration

_l2cap_data_receive_callback Function

File

l2cap_private.h

C

```
void _l2cap_data_receive_callback(bt_l2cap_mgr_p pmg, bt_hci_conn_state_p pconn, bt_byte_p pdata, bt_int
len);
```

Description

From l2cap_recv.c

_pack_cmd_reject Function

File

l2cap_private.h

C

```
bt_int _pack_cmd_reject(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p poffset);
```

Description

This is function _pack_cmd_reject.

_pack_config_request Function

File

l2cap_private.h

C

```
bt_int _pack_config_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_config_request.

_pack_config_response Function

File

l2cap_private.h

C

```
bt_int _pack_config_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_config_response.

_pack_conn_param_update_request Function

File

l2cap_private.h

C

```
bt_int _pack_conn_param_update_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len,
bt_int_p poffset);
```

Description

This is function _pack_conn_param_update_request.

_pack_conn_param_update_response Function

File

l2cap_private.h

C

```
bt_int _pack_conn_param_update_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len,
bt_int_p poffset);
```

Description

This is function _pack_conn_param_update_response.

_pack_conn_request Function

File

l2cap_private.h

C

```
bt_int _pack_conn_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_conn_request.

_pack_conn_response Function

File

l2cap_private.h

C

```
bt_int _pack_conn_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_conn_response.

_pack_dconn_request Function

File

l2cap_private.h

C

```
bt_int _pack_dconn_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_dconn_request.

_pack_dconn_response Function

File

l2cap_private.h

C

```
bt_int _pack_dconn_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_dconn_response.

_pack_echo_request Function

File

l2cap_private.h

C

```
bt_int _pack_echo_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_echo_request.

_pack_echo_response Function

File

l2cap_private.h

C

```
bt_int _pack_echo_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_echo_response.

_pack_info_request Function

File

l2cap_private.h

C

```
bt_int _pack_info_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_info_request.

_pack_info_response Function

File

l2cap_private.h

C

```
bt_int _pack_info_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_info_response.

_process_config_req Function

File

l2cap_private.h

C

```
void _process_config_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_config.c

_process_config_res Function

File

l2cap_private.h

C

```
void _process_config_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_config_res.

_process_conn_param_update_req Function

File

l2cap_private.h

C

```
void _process_conn_param_update_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_t*
pcmd);
```

Description

This is function _process_conn_param_update_req.

_process_conn_param_update_res Function

File

l2cap_private.h

C

```
void _process_conn_param_update_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_t* pcmd);
```

Description

This is function _process_conn_param_update_res.

_process_conn_req Function

File

l2cap_private.h

C

```
void _process_conn_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_connect.c

_process_conn_res Function

File

l2cap_private.h

C

```
void _process_conn_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_conn_res.

_process_dconn_req Function

File

l2cap_private.h

C

```
void _process_dconn_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_disconnect.c

_process_dconn_res Function

File

l2cap_private.h

C

```
void _process_dconn_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_dconn_res.

_process_echo_req Function

File

l2cap_private.h

C

```
void _process_echo_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_echo.c

_process_echo_res Function

File

l2cap_private.h

C

```
void _process_echo_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_echo_res.

_process_info_req Function

File

l2cap_private.h

C

```
void _process_info_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_info.c

_process_info_res Function

File

l2cap_private.h

C

```
void _process_info_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_info_res.

_process_reject Function

File

l2cap_private.h

C

```
void _process_reject(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_reject.c

_process_unknown_req Function

File

l2cap_private.h

C

```
void _process_unknown_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_unknown.c

_process_unknown_res Function

File

l2cap_private.h

C

```
void _process_unknown_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_unknown_res.

_read_cmd_reject Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_cmd_reject(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_cmd_reject.

_read_config_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_config_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_config_request.

_read_config_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_config_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_config_response.

_read_conn_param_update_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_param_update_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_param_update_request.

_read_conn_param_update_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_param_update_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_param_update_response.

_read_conn_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_request.

_read_conn_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_response.

_read_dconn_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_dconn_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_dconn_request.

_read_dconn_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_dconn_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_dconn_response.

_read_echo_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_echo_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_echo_request.

_read_echo_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_echo_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_echo_response.

_read_info_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_info_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_info_request.

_read_info_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_info_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_info_response.

_bt_l2cap_eretr_pack_config_request Function

File

l2cap_private.h

C

```
void _bt_l2cap_eretr_pack_config_request(bt_l2cap_channel_t* channel, bt_byte* buffer, bt_int buffer_len,
bt_int* offset);
```

Description

This is function _bt_l2cap_eretr_pack_config_request.

_bt_l2cap_eretr_recv Function

File

l2cap_eretr.h

C

```
void _bt_l2cap_eretr_recv(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte* pdata, bt_int len);
```

Description

This is function _bt_l2cap_eretr_recv.

_bt_l2cap_eretr_send_data Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_data(bt_l2cap_channel_t* pch, bt_byte* data, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function _bt_l2cap_eretr_send_data.

_bt_l2cap_eretr_send_pending_frames Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_pending_frames(bt_l2cap_channel_t* pch, bt_byte fbit);
```

Description

This is function _bt_l2cap_eretr_send_pending_frames.

_bt_l2cap_eretr_send_smart_data Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_smart_data(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function _bt_l2cap_eretr_send_smart_data.

_bt_l2cap_recv_req_seq_and_fbit Function

File

l2cap_eretr.h

C

```
void _bt_l2cap_recv_req_seq_and_fbit(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte req_seq, bt_byte fbit);
```

Description

This is function _bt_l2cap_recv_req_seq_and_fbit.

_bt_l2cap_send_ack Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_ack(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_ack.

_bt_l2cap_send_i_or_rr_or_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_i_or_rr_or_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_i_or_rr_or_rnr.

_bt_l2cap_send_rej Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rej(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rej.

_bt_l2cap_send_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rnr.

_bt_l2cap_send_rr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rr.

_bt_l2cap_send_rr_or_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rr_or_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rr_or_rnr.

bt_l2cap_disconnect_ex Function

File

l2cap.h

C

```
bt_bool bt_l2cap_disconnect_ex(bt_l2cap_channel_t* pch, bt_bool force_hci_disconnect);
```

Description

This is function bt_l2cap_disconnect_ex.

bt_l2cap_free_channel Function

File

chmanager.h

C

```
void bt_l2cap_free_channel(bt_l2cap_channel_t* ch);
```

Description

This is function bt_l2cap_free_channel.

bt_l2cap_hci_has_open_channels Function

File

l2cap.h

C

```
bt_bool bt_l2cap_hci_has_open_channels(bt_hci_conn_state_t* conn);
```

Description

This is function bt_l2cap_hci_has_open_channels.

bt_l2cap_is_channel_open Function

File

l2cap.h

C

```
bt_bool bt_l2cap_is_channel_open(bt_l2cap_channel_t* channel);
```

Description

This is function bt_l2cap_is_channel_open.

bt_l2cap_send_config Function

File

l2cap.h

C

```
bt_bool bt_l2cap_send_config(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch);
```

Description

This is function bt_l2cap_send_config.

bt_l2cap_send_smart_data Function

File

channel.h

C

```
bt_bool bt_l2cap_send_smart_data(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function bt_l2cap_send_smart_data.

_bt_l2cap_process_connect_signal Function

File

l2cap_private.h

C

```
void _bt_l2cap_process_connect_signal(bt_l2cap_mgr_t* mgr);
```

Description

This is function _bt_l2cap_process_connect_signal.

L2CAP Data Types and Constants

bt_l2cap_connect Macro

File

l2cap.h

C

```
#define bt_l2cap_connect(mgr, remote_addr, psm, acl_config, connect_cb, param, state_cb)
bt_l2cap_connect_ext(mgr, remote_addr, psm, CMODE_BASIC, acl_config, connect_cb, param, state_cb)
```

Description

This is macro bt_l2cap_connect.

bt_l2cap_listen Macro

File

l2cap.h

C

```
#define bt_l2cap_listen(mgr, psm, acl_config, callback, param) bt_l2cap_listen_ext(mgr, psm, CMODE_BASIC,
acl_config, callback, param)
```

Description

This is macro bt_l2cap_listen.

bt_l2cap_test_enable_local_config Macro

File

l2cap_test.h

C

```
#define bt_l2cap_test_enable_local_config(enable)
```

Description

This is macro bt_l2cap_test_enable_local_config.

bt_l2cap_test_enable_remote_config Macro

File

l2cap_test.h

C

```
#define bt_l2cap_test_enable_remote_config(enable)
```

Description

This is macro bt_l2cap_test_enable_remote_config.

CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED Macro

File

l2cap_private.h

C

```
#define CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED 0
```

Description

This is macro CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED.

CID_ATT Macro

File

channel.h

C

```
#define CID_ATT 0x0004
```

Description

This is macro CID_ATT.

CID_LE_SIG Macro

File

channel.h

C

```
#define CID_LE_SIG 0x0005
```

Description

This is macro CID_LE_SIG.

CID_MAX Macro

File

channel.h

C

```
#define CID_MAX 0xffff
```

Description

This is macro CID_MAX.

CID_MAX_FIXED Macro

File

channel.h

C

```
#define CID_MAX_FIXED 0x003f
```

Description

This is macro CID_MAX_FIXED.

CID_NULL Macro

File

channel.h

C

```
#define CID_NULL 0x0000
```

Description

This is macro CID_NULL.

CID_RECV Macro

File

channel.h

C

```
#define CID_RECV 0x0002
```

Description

This is macro CID_RECV.

CID_SIG Macro

File

channel.h

C

```
#define CID_SIG 0x0001
```

Description

This is macro CID_SIG.

CID_SM Macro

File

channel.h

C

```
#define CID_SM 0x0006
```

Description

This is macro CID_SM.

CMODE_BASIC Macro

File

channel.h

C

```
#define CMODE_BASIC 0
```

Description

This is macro CMODE_BASIC.

CMODE_ERETR Macro

File

channel.h

C

```
#define CMODE_ERETR 3
```

Description

This is macro CMODE_ERETR.

CMODE_FLOW Macro

File

channel.h

C

```
#define CMODE_FLOW 2
```

Description

This is macro CMODE_FLOW.

CMODE_RETR Macro

File

channel.h

C

```
#define CMODE_RETR 1
```

Description

This is macro CMODE_RETR.

CMODE_STRM Macro

File

channel.h

C

```
#define CMODE_STRM 4
```

Description

This is macro CMODE_STRM.

CSTATE_CLOSED Macro

File

channel.h

C

```
#define CSTATE_CLOSED 0x01
```

Description

This is macro CSTATE_CLOSED.

CSTATE_FREE Macro

File

channel.h

C

```
#define CSTATE_FREE 0x00
```

Description

This is macro CSTATE_FREE.

CSTATE_OPEN Macro

File

channel.h

C

```
#define CSTATE_OPEN 0x40
```

Description

This is macro CSTATE_OPEN.

CSTATE_WAIT_CONFIG Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG (CSTATE_WAIT_CONFIG_RSP | CSTATE_WAIT_CONFIG_REQ)
```

Description

This is macro CSTATE_WAIT_CONFIG.

CSTATE_WAIT_CONFIG_REQ Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG_REQ 0x10
```

Description

This is macro CSTATE_WAIT_CONFIG_REQ.

CSTATE_WAIT_CONFIG_RSP Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG_RSP 0x08
```

Description

This is macro CSTATE_WAIT_CONFIG_RSP.

CSTATE_WAIT_CONNECT Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONNECT 0x02
```

Description

This is macro CSTATE_WAIT_CONNECT.

CSTATE_WAIT_CONNECT_RSP Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONNECT_RSP 0x04
```

Description

This is macro CSTATE_WAIT_CONNECT_RSP.

CSTATE_WAIT_DISCONNECT Macro

File

channel.h

C

```
#define CSTATE_WAIT_DISCONNECT 0x20
```

Description

This is macro CSTATE_WAIT_DISCONNECT.

CTYPE_CL Macro

File

channel.h

C

```
#define CTYPE_CL 1 // connectionless bt_l2cap_channel
```

Description

connectionless bt_l2cap_channel

CTYPE_CO Macro

File

channel.h

C

```
#define CTYPE_CO 0 // connection-oriented bt_l2cap_channel
```

Description

connection-oriented bt_l2cap_channel

GET_F_BIT Macro

File

l2cap_eretr.h

C

```
#define GET_F_BIT(ctl) ((ctl & 0x80) >> 7)
```

Description

This is macro GET_F_BIT.

GET_FRAME_TYPE Macro

File

l2cap_eretr.h

C

```
#define GET_FRAME_TYPE(ctl) (ctl & 1)
```

Description

This is macro GET_FRAME_TYPE.

GET_P_BIT Macro

File

l2cap_eretr.h

C

```
#define GET_P_BIT(ctl) ((ctl & 0x10) >> 4)
```

Description

This is macro GET_P_BIT.

GET_REQ_SEQ Macro

File

l2cap_eretr.h

C

```
#define GET_REQ_SEQ(ctl) ((ctl & 0x3F00) >> 8)
```

Description

This is macro GET_REQ_SEQ.

GET_S_FUNCTION Macro

File

l2cap_eretr.h

C

```
#define GET_S_FUNCTION(ctl) ((ctl & 0x0C) >> 2)
```

Description

This is macro GET_S_FUNCTION.

GET_SAR Macro

File

l2cap_eretr.h

C

```
#define GET_SAR(ctl) ((ctl & 0xC000) >> 14)
```

Description

This is macro GET_SAR.

GET_TX_SEQ Macro

File

l2cap_eretr.h

C

```
#define GET_TX_SEQ(ctl) ((ctl & 0x7E) >> 1)
```

Description

This is macro GET_TX_SEQ.

L2CAP_CHANNEL_FLAG_INCOMING Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_INCOMING 0x02
```

Description

This is macro L2CAP_CHANNEL_FLAG_INCOMING.

L2CAP_CHANNEL_FLAG_SENDING Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_SENDING 0x01
```

Description

This is macro L2CAP_CHANNEL_FLAG_SENDING.

L2CAP_CMD_CONFIG_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONFIG_REQUEST 0x04
```

Description

This is macro L2CAP_CMD_CONFIG_REQUEST.

L2CAP_CMD_CONFIG_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONFIG_RESPONSE 0x05
```

Description

This is macro L2CAP_CMD_CONFIG_RESPONSE.

L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST 0x12
```

Description

This is macro L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST.

L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE 0x13
```

Description

This is macro L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE.

L2CAP_CMD_CONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_REQUEST 0x02
```

Description

This is macro L2CAP_CMD_CONN_REQUEST.

L2CAP_CMD_CONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_RESPONSE 0x03
```

Description

This is macro L2CAP_CMD_CONN_RESPONSE.

L2CAP_CMD_DATA_LEN_CMD_REJECT Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CMD_REJECT 2
```

Description

This is macro L2CAP_CMD_DATA_LEN_CMD_REJECT.

L2CAP_CMD_DATA_LEN_CONFIG_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONFIG_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONFIG_REQUEST.

L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE 6
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE.

L2CAP_CMD_DATA_LEN_CONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONN_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONN_REQUEST.

L2CAP_CMD_DATA_LEN_CONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONN_RESPONSE 8
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONN_RESPONSE.

L2CAP_CMD_DATA_LEN_DCONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_DCONN_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_DCONN_REQUEST.

L2CAP_CMD_DATA_LEN_DCONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_DCONN_RESPONSE 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_DCONN_RESPONSE.

L2CAP_CMD_DATA_LEN_ECHO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_ECHO_REQUEST 0
```

Description

This is macro L2CAP_CMD_DATA_LEN_ECHO_REQUEST.

L2CAP_CMD_DATA_LEN_ECHO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_ECHO_RESPONSE 0
```

Description

This is macro L2CAP_CMD_DATA_LEN_ECHO_RESPONSE.

L2CAP_CMD_DATA_LEN_INFO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_INFO_REQUEST 2
```

Description

This is macro L2CAP_CMD_DATA_LEN_INFO_REQUEST.

L2CAP_CMD_DATA_LEN_INFO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_INFO_RESPONSE 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_INFO_RESPONSE.

L2CAP_CMD_DCONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DCONN_REQUEST 0x06
```

Description

This is macro L2CAP_CMD_DCONN_REQUEST.

L2CAP_CMD_DCONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DCONN_RESPONSE 0x07
```

Description

This is macro L2CAP_CMD_DCONN_RESPONSE.

L2CAP_CMD_ECHO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_ECHO_REQUEST 0x08
```

Description

This is macro L2CAP_CMD_ECHO_REQUEST.

L2CAP_CMD_ECHO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_ECHO_RESPONSE 0x09
```

Description

This is macro L2CAP_CMD_ECHO_RESPONSE.

L2CAP_CMD_HEADER_LEN Macro

File

command.h

C

```
#define L2CAP_CMD_HEADER_LEN 4
```

Description

This is macro L2CAP_CMD_HEADER_LEN.

L2CAP_CMD_INFO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_INFO_REQUEST 0x0A
```

Description

This is macro L2CAP_CMD_INFO_REQUEST.

L2CAP_CMD_INFO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_INFO_RESPONSE 0x0B
```

Description

This is macro L2CAP_CMD_INFO_RESPONSE.

L2CAP_CMD_LAST Macro

File

command.h

C

```
#define L2CAP_CMD_LAST L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE
```

Description

This is macro L2CAP_CMD_LAST.

L2CAP_CMD_REJECT Macro

File

command.h

C

```
#define L2CAP_CMD_REJECT 0x01
```

Description

This is macro L2CAP_CMD_REJECT.

L2CAP_CMD_RESERVED Macro

File

command.h

C

```
#define L2CAP_CMD_RESERVED 0x00
```

Description

This is macro L2CAP_CMD_RESERVED.

L2CAP_CMD_STATUS_BEING_SENT Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_BEING_SENT 2
```

Description

This is macro L2CAP_CMD_STATUS_BEING_SENT.

L2CAP_CMD_STATUS_PENDING Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_PENDING 0
```

Description

This is macro L2CAP_CMD_STATUS_PENDING.

L2CAP_CMD_STATUS_WAITING_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.

L2CAP_CONFIG_RESULT_REJECTED Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_REJECTED 0x0002
```

Description

This is macro L2CAP_CONFIG_RESULT_REJECTED.

L2CAP_CONFIG_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_CONFIG_RESULT_SUCCESS.

L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER 0x0001
```

Description

This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.

L2CAP_CONFIG_RESULT_UNKNOWN_OPTION Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_UNKNOWN_OPTION 0x0003
```

Description

This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.

L2CAP_CONN_REQ_RESULT_INVALID_PSM Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_INVALID_PSM 0x0002
```

Description

This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.

L2CAP_CONN_REQ_RESULT_NO_RESOURCES Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_NO_RESOURCES 0x0004
```

Description

This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.

L2CAP_CONN_REQ_RESULT_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_PENDING 0x0001
```

Description

This is macro L2CAP_CONN_REQ_RESULT_PENDING.

L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK 0x0003
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.

L2CAP_CONN_REQ_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.

L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING 0x0001
```

Description

This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.

L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING 0x0002
```

Description

This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.

L2CAP_CONN_REQ_STATUS_NO_INFO Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_NO_INFO 0x0000
```

Description

This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.

L2CAP_DEFAULT_ERTX Macro

File

l2cap.h

C

```
#define L2CAP_DEFAULT_ERTX 60 /* seconds */
```

Description

seconds

L2CAP_DEFAULT_RTX Macro

File

l2cap.h

C

```
#define L2CAP_DEFAULT_RTX 3 /* seconds */
```

Description

seconds

L2CAP_ECHO_MAX_DATA_LEN Macro

File

command.h

C

```
#define L2CAP_ECHO_MAX_DATA_LEN 20
```

Description

echo request and response

L2CAP_ERETR_RECV_STATE_RECV Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_RECV 0
```

Description

This is macro L2CAP_ERETR_RECV_STATE_RECV.

L2CAP_ERETR_RECV_STATE_REJ_SENT Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_REJ_SENT 1
```

Description

This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.

L2CAP_ERETR_RECV_STATE_SREJ_SENT Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_SREJ_SENT 2
```

Description

This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.

L2CAP_ERETR_XMIT_STATE_WAIT_ACK Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_WAIT_ACK 1
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.

L2CAP_ERETR_XMIT_STATE_WAIT_F Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_WAIT_F 2
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.

L2CAP_ERETR_XMIT_STATE_XMIT Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_XMIT 0
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_XMIT.

L2CAP_EXT_BI_QOS Macro

File

command.h

C

```
#define L2CAP_EXT_BI_QOS 0x00000004
```

Description

This is macro L2CAP_EXT_BI_QOS.

L2CAP_EXT_ENHANCED_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_EXT_ENHANCED_RETRANSMISSION 0x00000008
```

Description

This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.

L2CAP_EXT_FCS_OPTION Macro

File

command.h

C

```
#define L2CAP_EXT_FCS_OPTION 0x00000020
```

Description

This is macro L2CAP_EXT_FCS_OPTION.

L2CAP_EXT_FLOW_CONTROL Macro

File

command.h

C

```
#define L2CAP_EXT_FLOW_CONTROL 0x00000001
```

Description

This is macro L2CAP_EXT_FLOW_CONTROL.

L2CAP_EXT_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_EXT_RETRANSMISSION 0x00000002
```

Description

This is macro L2CAP_EXT_RETRANSMISSION.

L2CAP_EXT_STREAMING Macro

File

command.h

C

```
#define L2CAP_EXT_STREAMING 0x00000010
```

Description

This is macro L2CAP_EXT_STREAMING.

L2CAP_FRAME_TYPE_I Macro

File

l2cap.h

C

```
#define L2CAP_FRAME_TYPE_I 0x0
```

Description

This is macro L2CAP_FRAME_TYPE_I.

L2CAP_FRAME_TYPE_S Macro

File

l2cap.h

C

```
#define L2CAP_FRAME_TYPE_S 0x1
```

Description

This is macro L2CAP_FRAME_TYPE_S.

L2CAP_HCI_CONNECT_PACKET_TYPE Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_PACKET_TYPE _l2cap_hci_connect_packet_type
```

Description

This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.

L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE _l2cap_hci_page_scan_repetition_mode
```

Description

This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.

L2CAP_HCI_CONNECT_ROLE_SWITCH Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_ROLE_SWITCH _l2cap_hci_role_switch
```

Description

This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.

L2CAP_HEADER_LEN Macro

File

l2cap.h

C

```
#define L2CAP_HEADER_LEN 4
```

Description

This is macro L2CAP_HEADER_LEN.

L2CAP_IDLE_HCI_CONNECTION_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_IDLE_HCI_CONNECTION_TIMEOUT _l2cap_idle_hci_connection_timeout
```

Description

This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.

L2CAP_INFO_NOT_SUPPORTED Macro

File

command.h

C

```
#define L2CAP_INFO_NOT_SUPPORTED 0x0000     // REDFLAG: should it be non zero?
```

Description

REDFLAG: should it be non zero?

L2CAP_INFO_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_INFO_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_INFO_RESULT_SUCCESS.

L2CAP_INFO_TYPE_CONNECTIONLESS_MTU Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_CONNECTIONLESS_MTU 0x0001
```

Description

information request

L2CAP_INFO_TYPE_EXTENDED_SUPPORT Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_EXTENDED_SUPPORT 0x0002
```

Description

This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.

L2CAP_LINK_TYPE_BD_EDR Macro

File

l2cap_fixed_channel.h

C

```
#define L2CAP_LINK_TYPE_BD_EDR HCI_LINK_TYPE_BD_EDR
```

Description

This is macro L2CAP_LINK_TYPE_BD_EDR.

L2CAP_LINK_TYPE_LE Macro

File

l2cap_fixed_channel.h

C

```
#define L2CAP_LINK_TYPE_LE HCI_LINK_TYPE_LE
```

Description

This is macro L2CAP_LINK_TYPE_LE.

L2CAP_MAX_CMD_QUEUE_LEN Macro

File

l2cap_command_queue.h

C

```
#define L2CAP_MAX_CMD_QUEUE_LEN 5
```

Description

This is macro L2CAP_MAX_CMD_QUEUE_LEN.

L2CAP_MAX_FRAME_BUFFERS Macro

File

frame_buffer.h

C

```
#define L2CAP_MAX_FRAME_BUFFERS 2
```

Description

This is macro L2CAP_MAX_FRAME_BUFFERS.

L2CAP_MAX_MANAGERS Macro

File

l2cap.h

C

```
#define L2CAP_MAX_MANAGERS 1
```

Description

This is macro L2CAP_MAX_MANAGERS.

L2CAP_MAX_MTU Macro

File

l2cap.h

C

```
#define L2CAP_MAX_MTU (_hci_l2cap_buffer_len - L2CAP_HEADER_LEN)
```

Description

This is macro L2CAP_MAX_MTU.

L2CAP_MAX_OPTIONS Macro

File

command.h

C

```
#define L2CAP_MAX_OPTIONS 4
```

Description

configuration request/response

L2CAP_MAX_RTX Macro

File

l2cap.h

C

```
#define L2CAP_MAX_RTX 4
```

Description

This is macro L2CAP_MAX_RTX.

L2CAP_MGR_STATE_FREE Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_FREE 0
```

Description

This is macro L2CAP_MGR_STATE_FREE.

L2CAP_MGR_STATE_LISTENING_L2CAP Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_LISTENING_L2CAP 2
```

Description

This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.

L2CAP_MGR_STATE_USED Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_USED 1
```

Description

This is macro L2CAP_MGR_STATE_USED.

L2CAP_MIN_MTU Macro

File

l2cap.h

C

```
#define L2CAP_MIN_MTU 48
```

Description

This is macro L2CAP_MIN_MTU.

L2CAP_MONITOR_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_MONITOR_TIMEOUT ((L2CAP_RFC_MONITOR_TIMEOUT / 1000) - 2)
```

Description

This is macro L2CAP_MONITOR_TIMEOUT.

L2CAP_OPTION_LEN_FLASH_QOS Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_QOS 22
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_QOS.

L2CAP_OPTION_LEN_FLASH_RFC Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_RFC 9
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_RFC.

L2CAP_OPTION_LEN_FLASH_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_TIMEOUT 2
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.

L2CAP_OPTION_LEN_MAX_MTU Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_MAX_MTU 2
```

Description

This is macro L2CAP_OPTION_LEN_MAX_MTU.

L2CAP_OPTION_LEN_UNKNOWN Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_UNKNOWN 22
```

Description

This is macro L2CAP_OPTION_LEN_UNKNOWN.

L2CAP_OPTION_LEN_UNKNOWN_DATA Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_UNKNOWN_DATA 9//21
```

Description

21

L2CAP_OPTION_TYPE_FLASH_QOS Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_QOS 0x03
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_QOS.

L2CAP_OPTION_TYPE_FLASH_QOS_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_QOS_FLAG 0x04
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.

L2CAP_OPTION_TYPE_FLASH_RFC Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_RFC 0x04
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_RFC.

L2CAP_OPTION_TYPE_FLASH_RFC_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_RFC_FLAG 0x08
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.

L2CAP_OPTION_TYPE_FLASH_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_TIMEOUT 0x02
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.

L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG 0x02
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.

L2CAP_OPTION_TYPE_MAX_MTU Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_MAX_MTU 0x01
```

Description

This is macro L2CAP_OPTION_TYPE_MAX_MTU.

L2CAP_OPTION_TYPE_MAX_MTU_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_MAX_MTU_FLAG 0x01
```

Description

This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.

L2CAP_OPTION_TYPE_UNKNOWN_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_UNKNOWN_FLAG 0x10
```

Description

This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.

L2CAP_PACKET_DATA_TYPE_RAW Macro

File

l2cap_packet.h

C

```
#define L2CAP_PACKET_DATA_TYPE_RAW 0
```

Description

This is macro L2CAP_PACKET_DATA_TYPE_RAW.

L2CAP_PACKET_DATA_TYPE_SMART Macro

File

l2cap_packet.h

C

```
#define L2CAP_PACKET_DATA_TYPE_SMART 1
```

Description

This is macro L2CAP_PACKET_DATA_TYPE_SMART.

L2CAP_REJECT_REASON_INVALID_CHANNEL Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_INVALID_CHANNEL 0x0002
```

Description

This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.

L2CAP_REJECT_REASON_MTU_EXCEEDED Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_MTU_EXCEEDED 0x0001
```

Description

This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.

L2CAP_REJECT_REASON_NOT_UNDERSTOOD Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_NOT_UNDERSTOOD 0x0000
```

Description

This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.

L2CAP_RETR_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_RETR_TIMEOUT (L2CAP_RFC_ERETR_TIMEOUT / 1000)
```

Description

This is macro L2CAP_RETR_TIMEOUT.

L2CAP_RFC_BASIC Macro

File

command.h

C

```
#define L2CAP_RFC_BASIC 0x00
```

Description

This is macro L2CAP_RFC_BASIC.

L2CAP_RFC_ENHANCED_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_RFC_ENHANCED_RETRANSMISSION 0x03
```

Description

This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.

L2CAP_RFC_ERETR_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_RFC_ERETR_TIMEOUT (3 * 1000)  // 3 secs
```

Description

3 secs

L2CAP_RFC_ERETR_TX_WINDOW Macro

File

command.h

C

```
#define L2CAP_RFC_ERETR_TX_WINDOW 1
```

Description

This is macro L2CAP_RFC_ERETR_TX_WINDOW.

L2CAP_RFC_FLOW_CONTROL Macro

File

command.h

C

```
#define L2CAP_RFC_FLOW_CONTROL 0x02
```

Description

This is macro L2CAP_RFC_FLOW_CONTROL.

L2CAP_RFC_MONITOR_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_RFC_MONITOR_TIMEOUT (12 * 1000) // 12 secs
```

Description

12 secs

L2CAP_RFC_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_RFC_RETRANSMISSION 0x01
```

Description

This is macro L2CAP_RFC_RETRANSMISSION.

L2CAP_RFC_STREAMING Macro

File

command.h

C

```
#define L2CAP_RFC_STREAMING 0x04
```

Description

This is macro L2CAP_RFC_STREAMING.

L2CAP_SAR_SDU_CONTINUE Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_CONTINUE 0x3
```

Description

This is macro L2CAP_SAR_SDU_CONTINUE.

L2CAP_SAR_SDU_START Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_START 0x1
```

Description

This is macro L2CAP_SAR_SDU_START.

L2CAP_SAR_SDU_STOP Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_STOP 0x2
```

Description

This is macro L2CAP_SAR_SDU_STOP.

L2CAP_SAR_UNSEGMENTED Macro

File

l2cap.h

C

```
#define L2CAP_SAR_UNSEGMENTED 0x0
```

Description

This is macro L2CAP_SAR_UNSEGMENTED.

L2CAP_SERVICE_TYPE_BEST_EFFORT Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_BEST_EFFORT 0x01
```

Description

This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.

L2CAP_SERVICE_TYPE_GUARANTEED Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_GUARANTEED 0x02
```

Description

This is macro L2CAP_SERVICE_TYPE_GUARANTEED.

L2CAP_SERVICE_TYPE_NO_TRAFFIC Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_NO_TRAFFIC 0x00
```

Description

This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.

L2CAP_SFUNCTION_REJ Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_REJ 0x1
```

Description

This is macro L2CAP_SFUNCTION_REJ.

L2CAP_SFUNCTION_RNR Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_RNR 0x2
```

Description

This is macro L2CAP_SFUNCTION_RNR.

L2CAP_SFUNCTION_RR Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_RR 0x0
```

Description

This is macro L2CAP_SFUNCTION_RR.

L2CAP_SFUNCTION_SREJ Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_SREJ 0x3
```

Description

This is macro L2CAP_SFUNCTION_SREJ.

PSM_ATT Macro

File

l2cap.h

C

```
#define PSM_ATT 0x001F
```

Description

This is macro PSM_ATT.

PSM_AVCTP Macro

File

l2cap.h

C

```
#define PSM_AVCTP 0x0017
```

Description

This is macro PSM_AVCTP.

PSM_AVCTP_Browsing Macro

File

l2cap.h

C

```
#define PSM_AVCTP_Browsing 0x001B
```

Description

This is macro PSM_AVCTP_Browsing.

PSM_AVDTP Macro

File

l2cap.h

C

```
#define PSM_AVDTP 0x0019
```

Description

This is macro PSM_AVDTP.

PSM_BNEP Macro

File

l2cap.h

C

```
#define PSM_BNEP 0x000F
```

Description

This is macro PSM_BNEP.

PSM_HID_Control Macro

File

l2cap.h

C

```
#define PSM_HID_Control 0x0011
```

Description

This is macro PSM_HID_Control.

PSM_HID_Interrupt Macro

File

l2cap.h

C

```
#define PSM_HID_Interrupt 0x0013
```

Description

This is macro PSM_HID_Interrupt.

PSM_RFCOMM Macro

File

l2cap.h

C

```
#define PSM_RFCOMM 0x0003
```

Description

This is macro PSM_RFCOMM.

PSM_SDP Macro

File

l2cap.h

C

```
#define PSM_SDP 0x0001
```

Description

This is macro PSM_SDP.

SET_F_BIT Macro

File

l2cap_eretr.h

C

```
#define SET_F_BIT(ctl, fbit) ctl = ((ctl & ~0x80) | ((fbit & 1) << 7))
```

Description

This is macro SET_F_BIT.

SET_FRAME_TYPE Macro

File

l2cap_eretr.h

C

```
#define SET_FRAME_TYPE(ctl, type) ctl = ((ctl & ~1) | (type & 1))
```

Description

This is macro SET_FRAME_TYPE.

SET_P_BIT Macro

File

l2cap_eretr.h

C

```
#define SET_P_BIT(ctl, pbit) ctl = ((ctl & ~0x10) | ((pbit & 1) << 4))
```

Description

This is macro SET_P_BIT.

SET_REQ_SEQ Macro

File

l2cap_eretr.h

C

```
#define SET_REQ_SEQ(ctl, seq) ctl = ((ctl & ~0x3F00) | ((seq & 0x3F) << 8))
```

Description

This is macro SET_REQ_SEQ.

SET_S_FUNCTION Macro

File

l2cap_eretr.h

C

```
#define SET_S_FUNCTION(ctl, s) ctl = ((ctl & ~0x0C) | ((s & 0x3) << 2))
```

Description

This is macro SET_S_FUNCTION.

SET_SAR Macro

File

l2cap_eretr.h

C

```
#define SET_SAR(ctl, sar) ctl = ((ctl & ~0xC000) | ((seq & 0x3) << 14))
```

Description

This is macro SET_SAR.

SET_TX_SEQ Macro

File

l2cap_eretr.h

C

```
#define SET_TX_SEQ(ctl, seq) ctl = ((ctl & ~0x7E) | ((seq & 0x3F) << 1))
```

Description

This is macro SET_TX_SEQ.

_bt_l2cap_start_monitor_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_monitor_timer(pch) pch->ext->monitor_timer_start_time = L2CAP_MONITOR_TIMEOUT
```

Description

This is macro _bt_l2cap_start_monitor_timer.

_bt_l2cap_start_monitor_timer_if_not_running Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_monitor_timer_if_not_running(pch) { if (pch->ext->monitor_timer_start_time == 0)
pch->ext->monitor_timer_start_time = L2CAP_MONITOR_TIMEOUT; }
```

Description

This is macro _bt_l2cap_start_monitor_timer_if_not_running.

_bt_l2cap_start_retr_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_retr_timer(pch) pch->ext->retr_timer_start_time = L2CAP_RETR_TIMEOUT
```

Description

This is macro _bt_l2cap_start_retr_timer.

_bt_l2cap_start_retr_timer_if_not_running Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_retr_timer_if_not_running(pch) { if (pch->ext->retr_timer_start_time == 0)
pch->ext->retr_timer_start_time = L2CAP_RETR_TIMEOUT; }
```

Description

This is macro _bt_l2cap_start_retr_timer_if_not_running.

_bt_l2cap_stop_monitor_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_stop_monitor_timer(pch) pch->ext->monitor_timer_start_time = 0
```

Description

This is macro _bt_l2cap_stop_monitor_timer.

_bt_l2cap_stop_retr_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_stop_retr_timer(pch) pch->ext->retr_timer_start_time = 0
```

Description

This is macro _bt_l2cap_stop_retr_timer.

_bt_l2cap_channel_t Structure

File

channel.h

C

```
struct _bt_l2cap_channel_t {
  bt_id cid;
  struct _bt_l2cap_mgr_s * l2cap_mgr;
  bt_byte mode;
  bt_byte type;
  bt_byte state;
  bt_byte flags;
  bt_id cid_destination;
  bt_int psm;
  bt_int cfg_try_cnt;
  bt_l2cap_read_data_callback_fp _read_data_cb;
  bt_l2cap_send_data_callback_fp _send_data_cb;
  void* _send_data_cb_param;
  bt_l2cap_state_changed_callback_fp _state_changed_cb;
  void* _state_changed_param;
  struct _bt_hci_conn_state_s * hci_conn;
  bt_l2cap_packet_t tx_packet;
  bt_uint response_config_options;
  bt_uint response_mtu;
  bt_int response_flash_timeout;
  bt_l2cap_option_rfc_t response_option_rfc;
  bt_byte response_option_unknown_type;
  bt_l2cap_option_unknown_t response_option_unknown;
  bt_uint request_config_options;
  bt_int request_mtu;
  bt_int request_flash_timeout;
  bt_l2cap_option_rfc_t request_option_rfc;
  bt_byte connect_cmd_id;
  bt_byte signal_command;
  bt_signal_t signal;
  bt_l2cap_channel_ext_t* ext;
};
```

Members

Members	Description
bt_uint response_config_options;	configuration

Description

This is record _bt_l2cap_channel_t.

bt_l2cap_cfg_option_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cfg_option_t {
  bt_byte type;
  union {
    bt_l2cap_option_max_mtu_t max_mtu;
    bt_l2cap_option_flash_timeout_t flash_timeout;
    bt_l2cap_option_rfc_t rfc;
    bt_l2cap_option_unknown_t unknown;
  } opt;
} bt_l2cap_cfg_option_t, * bt_l2cap_cfg_option_p;
```

Members

Members	Description
bt_l2cap_option_rfc_t rfc;	option_qos qos;

Description

This is type bt_l2cap_cfg_option_p.

bt_l2cap_cfg_option_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cfg_option_t {
  bt_byte type;
  union {
    bt_l2cap_option_max_mtu_t max_mtu;
    bt_l2cap_option_flash_timeout_t flash_timeout;
    bt_l2cap_option_rfc_t rfc;
    bt_l2cap_option_unknown_t unknown;
  } opt;
} bt_l2cap_cfg_option_t, * bt_l2cap_cfg_option_p;
```

Members

Members	Description
bt_l2cap_option_rfc_t rfc;	option_qos qos;

Description

This is type bt_l2cap_cfg_option_t.

bt_l2cap_channel_t Type

File

channel.h

C

```
typedef struct _bt_l2cap_channel_t bt_l2cap_channel_t;
```

Description

This is type bt_l2cap_channel_t.

bt_l2cap_cmd_assembler_fp Type

File

l2cap_private.h

C

```
typedef bt_bool (* bt_l2cap_cmd_assembler_fp)(bt_l2cap_cmd_header_t* pcmd, bt_byte* buffer, bt_int
buffer_len, bt_int* poffset);
```

Description

This is type bt_l2cap_cmd_assembler_fp.

bt_l2cap_cmd_config_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_int flags;
  bt_int option_count;
```

```
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_req_t, * bt_l2cap_cmd_config_req_p;
```

Description

This is type bt_l2cap_cmd_config_req_p.

bt_l2cap_cmd_config_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_int flags;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_req_t, * bt_l2cap_cmd_config_req_p;
```

Description

This is type bt_l2cap_cmd_config_req_t.

bt_l2cap_cmd_config_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_source;
  bt_int flags;
  bt_int result;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_res_t, * bt_l2cap_cmd_config_res_p;
```

Description

This is type bt_l2cap_cmd_config_res_p.

bt_l2cap_cmd_config_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_source;
  bt_int flags;
  bt_int result;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_res_t, * bt_l2cap_cmd_config_res_p;
```

Description

This is type bt_l2cap_cmd_config_res_t.

bt_l2cap_cmd_conn_param_update_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_conn_param_update_req_t {
  bt_l2cap_cmd_header_t header;
  bt_uint min_interval;
  bt_uint max_interval;
  bt_uint slave_latency;
  bt_uint timeout_multiplier;
} bt_l2cap_cmd_conn_param_update_req_t;
```

Description

connection parameter update

bt_l2cap_cmd_conn_param_update_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_conn_param_update_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int result;
} bt_l2cap_cmd_conn_param_update_res_t;
```

Description

This is type bt_l2cap_cmd_conn_param_update_res_t.

bt_l2cap_cmd_connection_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int psm;
  bt_id cid_source;
} bt_l2cap_cmd_connection_req_t, * bt_l2cap_cmd_connection_req_p;
```

Description

This is type bt_l2cap_cmd_connection_req_p.

bt_l2cap_cmd_connection_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int psm;
  bt_id cid_source;
} bt_l2cap_cmd_connection_req_t, * bt_l2cap_cmd_connection_req_p;
```

Description

This is type bt_l2cap_cmd_connection_req_t.

bt_l2cap_cmd_connection_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
  bt_int result;
  bt_int status;
} bt_l2cap_cmd_connection_res_t, * bt_l2cap_cmd_connection_res_p;
```

Description

This is type bt_l2cap_cmd_connection_res_p.

bt_l2cap_cmd_connection_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
  bt_int result;
  bt_int status;
} bt_l2cap_cmd_connection_res_t, * bt_l2cap_cmd_connection_res_p;
```

Description

This is type bt_l2cap_cmd_connection_res_t.

bt_l2cap_cmd_disconnection_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_disconnection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} bt_l2cap_cmd_disconnection_req_t, * bt_l2cap_cmd_disconnection_req_p;
```

Description

This is type bt_l2cap_cmd_disconnection_req_p.

bt_l2cap_cmd_disconnection_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_disconnection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} bt_l2cap_cmd_disconnection_req_t, * bt_l2cap_cmd_disconnection_req_p;
```

Description

This is type bt_l2cap_cmd_disconnection_req_t.

bt_l2cap_cmd_echo_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_req_t, * bt_l2cap_cmd_echo_req_p;
```

Description

This is type bt_l2cap_cmd_echo_req_p.

bt_l2cap_cmd_echo_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_req_t, * bt_l2cap_cmd_echo_req_p;
```

Description

This is type bt_l2cap_cmd_echo_req_t.

bt_l2cap_cmd_echo_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_res_t, * bt_l2cap_cmd_echo_res_p;
```

Description

This is type bt_l2cap_cmd_echo_res_p.

bt_l2cap_cmd_echo_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_res_t, * bt_l2cap_cmd_echo_res_p;
```

Description

This is type bt_l2cap_cmd_echo_res_t.

bt_l2cap_cmd_header_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_header_t {
  struct _bt_l2cap_cmd_header_t* next_cmd;
  bt_byte code;
  bt_byte id;
  bt_byte status;
  bt_long send_time;
  bt_int rtx;
  bt_byte rtx_count;
  pf_l2cap_cmd_callback cb;
  struct _bt_hci_conn_state_s * phci_conn;
} bt_l2cap_cmd_header_t, * bt_l2cap_cmd_header_p;
```

Description

This is type bt_l2cap_cmd_header_p.

bt_l2cap_cmd_header_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_header_t {
  struct _bt_l2cap_cmd_header_t* next_cmd;
  bt_byte code;
  bt_byte id;
  bt_byte status;
  bt_long send_time;
  bt_int rtx;
  bt_byte rtx_count;
  pf_l2cap_cmd_callback cb;
  struct _bt_hci_conn_state_s * phci_conn;
} bt_l2cap_cmd_header_t, * bt_l2cap_cmd_header_p;
```

Description

This is type bt_l2cap_cmd_header_t.

bt_l2cap_cmd_info_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
} bt_l2cap_cmd_info_req_t, * bt_l2cap_cmd_info_req_p;
```

Description

This is type bt_l2cap_cmd_info_req_p.

bt_l2cap_cmd_info_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
} bt_l2cap_cmd_info_req_t, * bt_l2cap_cmd_info_req_p;
```

Description

This is type bt_l2cap_cmd_info_req_t.

bt_l2cap_cmd_info_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
  bt_int result;
  union {
    bt_int mtu;
    bt_long mask;
    bt_long fixed_channles[2];
  } data;
} bt_l2cap_cmd_info_res_t, * bt_l2cap_cmd_info_res_p;
```

Description

This is type bt_l2cap_cmd_info_res_p.

bt_l2cap_cmd_info_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
  bt_int result;
  union {
    bt_int mtu;
    bt_long mask;
    bt_long fixed_channles[2];
  } data;
} bt_l2cap_cmd_info_res_t, * bt_l2cap_cmd_info_res_p;
```

Description

This is type bt_l2cap_cmd_info_res_t.

bt_l2cap_cmd_parser_fp Type

File

l2cap_private.h

C

```
typedef bt_l2cap_cmd_header_t* (* bt_l2cap_cmd_parser_fp)(bt_byte* pdata, bt_int len, bt_int* poffset);
```

Description

This is type bt_l2cap_cmd_parser_fp.

bt_l2cap_cmd_reject_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_reject_t {
  bt_l2cap_cmd_header_t header;
  bt_uint reason;
  union {
    bt_int actual_mtu;
    struct {
      bt_id cid_local;
      bt_id cid_remote;
    } cid;
  } data;
} bt_l2cap_cmd_reject_t, * bt_l2cap_cmd_reject_p;
```

Description

This is type bt_l2cap_cmd_reject_p.

bt_l2cap_cmd_reject_param_t Union

File

command.h

C

```
typedef union _bt_l2cap_cmd_reject_param_t {
  bt_int actual_mtu;
  struct {
    bt_id cid_local;
    bt_id cid_remote;
  } cid;
} bt_l2cap_cmd_reject_param_t;
```

Description

This is type bt_l2cap_cmd_reject_param_t.

bt_l2cap_cmd_reject_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_reject_t {
  bt_l2cap_cmd_header_t header;
  bt_uint reason;
  union {
    bt_int actual_mtu;
    struct {
      bt_id cid_local;
      bt_id cid_remote;
    } cid;
  } data;
} bt_l2cap_cmd_reject_t, * bt_l2cap_cmd_reject_p;
```

Description

This is type bt_l2cap_cmd_reject_t.

bt_l2cap_command_t Union

File

command.h

C

```
typedef union _bt_l2cap_command_t {
  bt_l2cap_cmd_reject_t reject;
  bt_l2cap_cmd_connection_req_t connection_req;
  bt_l2cap_cmd_connection_res_t connection_res;
  bt_l2cap_cmd_disconnection_req_t disconnection_req;
  cmd_disconnection_res disconnection_res;
  bt_l2cap_cmd_config_req_t config_req;
  bt_l2cap_cmd_config_res_t config_res;
  bt_l2cap_cmd_echo_req_t echo_req;
  bt_l2cap_cmd_echo_res_t echo_res;
  bt_l2cap_cmd_info_req_t info_req;
  bt_l2cap_cmd_info_res_t info_res;
  bt_l2cap_cmd_conn_param_update_req_t conn_update_req;
  bt_l2cap_cmd_conn_param_update_res_t conn_update_res;
} bt_l2cap_command_t;
```

Description

This is type bt_l2cap_command_t.

bt_l2cap_connect_callback_fp Type

File

l2cap.h

C

```
typedef void (* bt_l2cap_connect_callback_fp)(bt_l2cap_channel_t* pch, void* param);
```

Description

This is type bt_l2cap_connect_callback_fp.

bt_l2cap_eretr_xmit_event_e Enumeration

File

l2cap_eretr.h

C

```
typedef enum _bt_l2cap_eretr_xmit_event_e {
  L2CAP_ERETR_EVENT_DATA_REQUEST,
  L2CAP_ERETR_EVENT_LOCAL_BUSY_DETECTED,
  L2CAP_ERETR_EVENT_LOCAL_BUSY_CLEAR,
  L2CAP_ERETR_EVENT_RECV_REQSEQ_AND_FBIT,
  L2CAP_ERETR_EVENT_RECV_FBIT,
  L2CAP_ERETR_EVENT_RETRANSMIT_TIMER_EXPIRED,
  L2CAP_ERETR_EVENT_MONITOR_TIMER_EXPIRED
} bt_l2cap_eretr_xmit_event_e;
```

Description

This is type bt_l2cap_eretr_xmit_event_e.

bt_l2cap_fixed_channel_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_fixed_channel_s {
```

```
  bt_id cid;
  bt_byte link_type;
  bt_l2cap_listen_callback_fp listen_cb;
  bt_l2cap_connect_callback_fp connect_cb;
  void* listen_cb_param;
  void* connect_cb_param;
} bt_l2cap_fixed_channel_t;
```

Description

This is type bt_l2cap_fixed_channel_t.

bt_l2cap_frame_desc_t Structure

File

channel.h

C

```
typedef struct _bt_l2cap_frame_desc_s {
  bt_uint ctl;
  bt_byte data_type;
  union {
    bt_byte* raw;
    bt_packet_t* packet;
  } data;
  bt_int len;
  bt_byte tx_seq;
  bt_byte retry_count;
  bt_uint fcs;
  bt_l2cap_send_data_callback_fp callback;
  void* callback_param;
} bt_l2cap_frame_desc_t;
```

Description

typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;

bt_l2cap_listen_callback_fp Type

File

l2cap.h

C

```
typedef void (* bt_l2cap_listen_callback_fp)(bt_l2cap_channel_t* pch, void* param);
```

Description

This is type bt_l2cap_listen_callback_fp.

bt_l2cap_mgr_p Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_mgr_s {
  bt_int state;
  bt_l2cap_psm_t* _psms;
  bt_l2cap_fixed_channel_t* _fixed_channels;
  bt_l2cap_channel_t* _channels;
  bt_queue_element_t* cmd_queue;
  bt_queue_element_t* cmd_res_queue;
  bt_byte max_rtx;
  bt_hci_ctrl_state_t* hci_ctrl;
  bt_l2cap_packet_t cmd_tx_packet;
  bt_queue_element_t* connect_queue;
  bt_buffer_mgr_t connect_params_mgr;
```

```
  struct {
    unsigned int connecting : 1;
    unsigned int sig_cmd : 1;
    unsigned int sig_connect : 1;
    unsigned int sending : 1;
  } flags;
  bt_hci_listener_t hci_disconnect_listener;
  bt_byte connect_chmode;
} bt_l2cap_mgr_t, * bt_l2cap_mgr_p;
```

Members

Members	Description
bt_queue_element_t* connect_queue;	bt_int connect_psm; bt_l2cap_connect_callback_fp connect_cb; void* connect_param; bt_l2cap_state_changed_callback_fp connect_state_cb;

Description

This is type bt_l2cap_mgr_p.

bt_l2cap_mgr_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_mgr_s {
  bt_int state;
  bt_l2cap_psm_t* _psms;
  bt_l2cap_fixed_channel_t* _fixed_channels;
  bt_l2cap_channel_t* _channels;
  bt_queue_element_t* cmd_queue;
  bt_queue_element_t* cmd_res_queue;
  bt_byte max_rtx;
  bt_hci_ctrl_state_t* hci_ctrl;
  bt_l2cap_packet_t cmd_tx_packet;
  bt_queue_element_t* connect_queue;
  bt_buffer_mgr_t connect_params_mgr;
  struct {
    unsigned int connecting : 1;
    unsigned int sig_cmd : 1;
    unsigned int sig_connect : 1;
    unsigned int sending : 1;
  } flags;
  bt_hci_listener_t hci_disconnect_listener;
  bt_byte connect_chmode;
} bt_l2cap_mgr_t, * bt_l2cap_mgr_p;
```

Members

Members	Description
bt_queue_element_t* connect_queue;	bt_int connect_psm; bt_l2cap_connect_callback_fp connect_cb; void* connect_param; bt_l2cap_state_changed_callback_fp connect_state_cb;

Description

This is type bt_l2cap_mgr_t.

bt_l2cap_option_flash_timeout_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_flash_timeout_t {
  bt_int timeout;
} bt_l2cap_option_flash_timeout_t;
```

Description

This is type bt_l2cap_option_flash_timeout_t.

bt_l2cap_option_max_mtu_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_max_mtu_t {
  bt_uint mtu;
} bt_l2cap_option_max_mtu_t;
```

Description

This is type bt_l2cap_option_max_mtu_t.

bt_l2cap_option_qos_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_qos_t {
  bt_byte flags;
  bt_byte service_type;
  bt_long token_rate;
  bt_long token_bucket_size;
  bt_long peak_bandwidth;
  bt_long latency;
  bt_long delay_variation;
} bt_l2cap_option_qos_t;
```

Description

This is type bt_l2cap_option_qos_t.

bt_l2cap_option_rfc_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_rfc_t {
  bt_byte mode;
  bt_byte tx_window_size;
  bt_byte max_transmit;
  bt_int retr_timeout;
  bt_int monitor_timeout;
  bt_int mps;
} bt_l2cap_option_rfc_t;
```

Description

This is type bt_l2cap_option_rfc_t.

bt_l2cap_option_unknown_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_unknown_t {
  bt_byte data_len;
  bt_byte data[L2CAP_OPTION_LEN_UNKNOWN_DATA];
} bt_l2cap_option_unknown_t;
```

Description

This is type bt_l2cap_option_unknown_t.

bt_l2cap_packet_t Structure

File

l2cap_packet.h

C

```
typedef struct _bt_l2cap_packet_t {
  bt_packet_t header;
  union {
    struct _bt_l2cap_channel_t* channel;
    bt_hci_hconn_t hconn;
  } destination;
  bt_byte data_type;
} bt_l2cap_packet_t;
```

Members

Members	Description
bt_byte data_type;	bt_uint fcs; bt_int ctl;

Description

This is type bt_l2cap_packet_t.

bt_l2cap_psm_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_psm_s {
  bt_int psm;
  bt_uint acl_config;
  bt_byte chmode;
  bt_l2cap_listen_callback_fp listen_cb;
  bt_l2cap_connect_callback_fp connect_cb;
  void* listen_cb_param;
  void* connect_cb_param;
} bt_l2cap_psm_t;
```

Description

This is type bt_l2cap_psm_t.

bt_l2cap_read_data_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_read_data_callback_fp)(bt_l2cap_channel_t* pch, bt_byte_p pdata, bt_int len);
```

Description

This is type bt_l2cap_read_data_callback_fp.

bt_l2cap_request_handler_fp Type

File

l2cap_private.h

C

```
typedef void (* bt_l2cap_request_handler_fp)(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn,
bt_l2cap_cmd_header_p pcmd);
```

Description

This is type bt_l2cap_request_handler_fp.

bt_l2cap_response_handler_fp Type

File

l2cap_private.h

C

```
typedef void (* bt_l2cap_response_handler_fp)(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is type bt_l2cap_response_handler_fp.

bt_l2cap_send_data_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_send_data_callback_fp)(bt_l2cap_channel_t* pch, bt_byte_p pdata, bt_int len, void*
param);
```

Description

This is type bt_l2cap_send_data_callback_fp.

bt_l2cap_state_changed_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_state_changed_callback_fp)(bt_l2cap_channel_t* pch, bt_int new_state, void* param);
```

Description

This is type bt_l2cap_state_changed_callback_fp.

bt_l2cap_xmit_event_param_t Structure

File

l2cap_eretr.h

C

```
typedef struct _bt_l2cap_xmit_event_param_t {
  bt_l2cap_eretr_xmit_event_e ev;
  bt_l2cap_mgr_p pmgr;
  bt_l2cap_channel_t* pch;
  union {
    struct {
      bt_byte req_seq;
      bt_byte fbit;
    } recv_reqseq_and_fbit;
    struct {
      bt_byte fbit;
    } recv_bfit;
    struct {
      bt_l2cap_frame_desc_t* frame;
```

```
    } data_request;
  } params;
} bt_l2cap_xmit_event_param_t;
```

Description

This is type bt_l2cap_xmit_event_param_t.

cmd_disconnection_res Structure

File

command.h

C

```
typedef struct _cmd_disconnection_res {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} cmd_disconnection_res, * pcmd_disconnection_res;
```

Description

This is type cmd_disconnection_res.

pcmd_disconnection_res Structure

File

command.h

C

```
typedef struct _cmd_disconnection_res {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} cmd_disconnection_res, * pcmd_disconnection_res;
```

Description

This is type pcmd_disconnection_res.

pf_l2cap_cmd_callback Type

File

command.h

C

```
typedef void (* pf_l2cap_cmd_callback)(struct _bt_l2cap_mgr_s *ppmg, struct _bt_l2cap_cmd_header_t *pcmd);
```

Description

This is type pf_l2cap_cmd_callback.

L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT 0x04
```

Description

This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.

L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR Macro

File

command.h

C

```
#define L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR 0x00000040
```

Description

This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.

L2CAP_EXT_EXT_WINDOW_SIZE Macro

File

command.h

C

```
#define L2CAP_EXT_EXT_WINDOW_SIZE 0x00000100
```

Description

This is macro L2CAP_EXT_EXT_WINDOW_SIZE.

L2CAP_EXT_FEATURES_ENABLED Macro

File

l2cap_private.h

C

```
#define L2CAP_EXT_FEATURES_ENABLED (_l2cap_eretr_recv_fp != NULL)
```

Description

This is macro L2CAP_EXT_FEATURES_ENABLED.

L2CAP_EXT_FIXED_CHANNELS Macro

File

command.h

C

```
#define L2CAP_EXT_FIXED_CHANNELS 0x00000080
```

Description

This is macro L2CAP_EXT_FIXED_CHANNELS.

L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT Macro

File

command.h

C

```
#define L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT 0x00000200
```

Description

This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.

L2CAP_INFO_TYPE_FIXED_CHANNELS Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_FIXED_CHANNELS 0x0003
```

Description

This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.

bt_l2cap_channel_ext_t Type

File

channel.h

C

```
typedef struct _bt_l2cap_channel_ext_t bt_l2cap_channel_ext_t;
```

Description

This is type bt_l2cap_channel_ext_t.

_bt_l2cap_channel_ext_t Structure

File

channel.h

C

```
struct _bt_l2cap_channel_ext_t {
  bt_bool check_fcs;
  bt_byte max_transmit;
  bt_byte next_tx_seq;
  bt_byte expected_ack_seq;
  bt_byte expected_tx_seq;
  bt_bool remote_busy;
  bt_bool local_busy;
  bt_l2cap_frame_desc_t unacked_frame;
  bt_l2cap_frame_desc_t pending_frame;
  bt_l2cap_frame_desc_t s_frame;
  bt_byte retry_count;
  bt_bool rnr_sent;
  bt_bool rej_actioned;
  bt_byte frames_sent;
  bt_ulong retr_timer_start_time;
  bt_ulong monitor_timer_start_time;
  bt_byte recv_state;
  bt_byte xmit_state;
  bt_l2cap_packet_t tx_s_packet;
};
```

Members

Members	Description
bt_byte expected_tx_seq;	bt_byte req_seq;
bt_bool remote_busy;	bt_byte buffer_seq;
bt_byte retry_count;	bt_byte srej_list[RFC_ERETR_TXWINDOW];
bt_byte frames_sent;	bt_bool srej_actioned; bt_byte srej_save_req_seq; bt_bool send_srej; bt_byte buffer_seq_srej;

Description

This is record _bt_l2cap_channel_ext_t.

_bt_l2cap_connect_params_s Structure

File

l2cap.h

C

```
struct _bt_l2cap_connect_params_s {
  bt_l2cap_connect_params_t* next;
  bt_bdaddr_t addr;
  bt_int psm;
  bt_l2cap_connect_callback_fp callback;
  void* param;
  bt_l2cap_state_changed_callback_fp state_cb;
  bt_byte chmode;
  bt_bool fixed_channel;
  bt_uint acl_config;
};
```

Description

This is record _bt_l2cap_connect_params_s.

bt_l2cap_connect_params_t Type

File

l2cap.h

C

```
typedef struct _bt_l2cap_connect_params_s bt_l2cap_connect_params_t;
```

Description

This is type bt_l2cap_connect_params_t.

L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID 0x0006
```

Description

This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.

L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED 0x0007
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.

Platform Data Types and Constants

BYTE_SIZE Macro

File

types.h

C

```
#define BYTE_SIZE 1
```

Description

This is macro BYTE_SIZE.

bt_byte Type

File

types.h

C

```
typedef unsigned char bt_byte;
```

Description

This is type bt_byte.

bt_int Type

File

types.h

C

```
typedef short bt_int;
```

Description

This is type bt_int.

bt_long Type

File

types.h

C

```
typedef long bt_long;
```

Description

This is type bt_long.

bt_uint Type

File

types.h

C

```
typedef unsigned short bt_uint;
```

Description

This is type bt_uint.

bt_ulong Type

File

types.h

C

```
typedef unsigned long bt_ulong;
```

Description

This is type bt_ulong.

RFCOMM Functions

bt_rfcomm_allocate_dlc Function

File

rfcomm.h

C

```
bt_rfcomm_dlc_t* bt_rfcomm_allocate_dlc(bt_rfcomm_session_t* session, bt_byte dlci);
```

Description

brief Allocate DLC. ingroup rfcomm

details This function allocates a new DLC on the specified RFCOMM session.

param session The RFCOMM session. param dlci DLCI of the new DLC.

return li A pointer to the new DLC if the function succeeds. li c NULL otherwise.

bt_rfcomm_allocate_session Function

File

rfcomm.h

C

```
bt_rfcomm_session_t* bt_rfcomm_allocate_session(bt_l2cap_mgr_t* l2cap_mgr);
```

Description

brief Allocate RFCOMM session. ingroup rfcomm

details This function allocates a new RFCOMM session.

param l2cap_mgr The L2CAP manager on which the RFCOMM session is to be created. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li A pointer to the new RFCOMM session structure if the function succeeds. li c NULL otherwise.

bt_rfcomm_close_dlc Function

File

rfcomm.h

C

```
void bt_rfcomm_close_dlc(bt_rfcomm_dlc_t* dlc);
```

Description

brief Close DLC. ingroup rfcomm

details This function closes a DLC. If DLCI = 0, the parent RFCOMM session is also closed.

bt_rfcomm_connect Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_connect(bt_bdaddr_p remote_addr, bt_byte server_channel, bt_uint acl_config,
bt_rfcomm_dlc_state_callback_fp callback, void* param);
```

Description

brief Connect to a remote device. ingroup rfcomm

details This function establishes an RFCOMM connection with a remote device and opens a data DLC. Changes in the session state are reported through a callback function specified when the session has been allocated via call to bt_rfcomm_allocate_session. Changes in the data DLC are reported through a callback function specified in this call.

param remote_addr Address of the remote device. param server_channel A server channel of the remote RFCOMM server. param callback The callback function for reporting changes in DLC state opened by this call. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_find_dlc Function

File

rfcomm.h

C

```
bt_rfcomm_dlc_t* bt_rfcomm_find_dlc(bt_rfcomm_session_t* session, bt_byte address);
```

Description

brief Find DLC

bt_rfcomm_free_dlc Function

File

rfcomm.h

C

```
void bt_rfcomm_free_dlc(bt_rfcomm_dlc_t* dlc);
```

Description

brief Release DLC. ingroup rfcomm

details This function releases the specified DLC.

param dlc The DLC to be released.

bt_rfcomm_free_session Function

File

rfcomm.h

C

```
void bt_rfcomm_free_session(bt_rfcomm_session_t* session);
```

Description

brief Release RFCOMM session. ingroup rfcomm

details This function deallocates the specified RFCOMM session. This function does not disconnect the session. It just frees the memory used by the bt_rfcomm_session structure. The session has to be disconnected by calling bt_rfcomm_close_dlc with DLCI = 0 first.

param session The RFCOMM session to be deallocated.

bt_rfcomm_init Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_init();
```

Description

brief Initialize the RFCOMM layer. ingroup rfcomm

details This function initializes the RFCOMM layer of the stack. It must be called prior to any other RFCOMM function can be called.

bt_rfcomm_listen Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_listen(bt_byte server_channel, bt_uint acl_config, bt_rfcomm_dlc_state_callback_fp
callback, void* param);
```

Description

brief Listen for incoming connections. ingroup rfcomm

details This function enables incoming connections on the specified RFCOMM session. Changes in the session state are reported through a callback function.

param server_channel A server channel on which the RFCOMM session will listen and accept incoming connections. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_open_dlc Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_open_dlc(bt_rfcomm_dlc_t* dlc, bt_rfcomm_dlc_state_callback_fp callback, void* param);
```

Description

brief Open DLC. ingroup rfcomm

details This function opens the specified DLC. Before calling this function the RFCOMM session must be already open. This function is not to be used with DLCI = 0. Changes in DLC state are reported through a callback function.

param dlc The DLC to open. param callback The callback function for reporting changes in DLC state. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_register_listener Function

File

rfcomm_mgr.h

C

```
bt_bool bt_rfcomm_register_listener(bt_rfcomm_session_listener_t* listener);
```

Description

This is function bt_rfcomm_register_listener.

bt_rfcomm_send_credit Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_send_credit(bt_rfcomm_dlc_t* dlc, bt_byte credit);
```

Description

This is function bt_rfcomm_send_credit.

bt_rfcomm_send_data Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_send_data(bt_rfcomm_dlc_t* dlc, void * data, bt_int len, bt_rfcomm_send_data_callback_fp
callback);
```

Description

brief Send data over a DLC. ingroup rfcomm

details This function sends data over the specified DLC. Operation completion is reported through callback function.

param dlc The DLC. param data A pointer to the data to be sent. param len Data length. param callback The callback function that is called when operation completes.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_unregister_listener Function

File

rfcomm_mgr.h

C

```
void bt_rfcomm_unregister_listener(bt_rfcomm_session_listener_t* listener);
```

Description

This is function bt_rfcomm_unregister_listener.

check_fcs Function

File

rfcomm_private.h

C

```
bt_bool check_fcs(bt_byte_p buffer, bt_byte len, bt_byte recv_fcs);
```

Description

From rfcomm_fcs.c

read_command Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p read_command(bt_byte_p pdata, bt_int len, bt_int* offset, bt_l2cap_cmd_header_t*
pheader);
```

Description

From channel_cmd_recv.c

rfcomm_cq_ack_cmd Function

File

rfcomm_cmd_queue.h

C

```
struct _bt_rfcomm_command_t* rfcomm_cq_ack_cmd(struct _bt_rfcomm_dlc_t* pdlc, struct _bt_rfcomm_command_t*
pres);
```

Description

This is function rfcomm_cq_ack_cmd.

rfcomm_cq_ack_mx_cmd Function

File

rfcomm_cmd_queue.h

C

```
void rfcomm_cq_ack_mx_cmd(struct _bt_rfcomm_dlc_t* pdlc, struct _bt_rfcomm_command_t* pres);
```

Description

This is function rfcomm_cq_ack_mx_cmd.

rfcomm_send_cmd Function

File

rfcomm_private.h

C

```
bt_bool rfcomm_send_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

This is function rfcomm_send_cmd.

rfcomm_send_commands_from_queue Function

File

rfcomm_cmd_queue.h

C

```
void rfcomm_send_commands_from_queue(struct _bt_rfcomm_session_t * psess);
```

Description

This is function rfcomm_send_commands_from_queue.

rfcomm_send_mx_msc_cmd Function

File

rfcomm_mx.h

C

```
bt_bool rfcomm_send_mx_msc_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function rfcomm_send_mx_msc_cmd.

rfcomm_send_mx_pn_cmd Function

File

rfcomm_mx.h

C

```
void rfcomm_send_mx_pn_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function rfcomm_send_mx_pn_cmd.

_rfcomm_alloc_cmd_buffer Function

File

rfcomm_private.h

C

```
bt_rfcomm_command_p _rfcomm_alloc_cmd_buffer();
```

Description

This is function _rfcomm_alloc_cmd_buffer.

_rfcomm_allocate_buffers Function

File

rfcomm_private.h

C

```
void _rfcomm_allocate_buffers();
```

Description

Defined by OEM through library configuration

_rfcomm_allocate_mx_cmd Function

File

rfcomm_mx.h

C

```
bt_rfcomm_command_p _rfcomm_allocate_mx_cmd(bt_rfcomm_session_p psess, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function _rfcomm_allocate_mx_cmd.

_rfcomm_free_cmd_buffer Function

File

rfcomm_private.h

C

```
void _rfcomm_free_cmd_buffer(void* p);
```

Description

This is function _rfcomm_free_cmd_buffer.

_rfcomm_get_tick_count Function

File

rfcomm_timer.h

C

```
bt_long _rfcomm_get_tick_count();
```

Description

This is function _rfcomm_get_tick_count.

_rfcomm_init_cmd_buffers Function

File

rfcomm_private.h

C

```
bt_bool _rfcomm_init_cmd_buffers();
```

Description

From rfcomm_cmdbuffer.c

_rfcomm_init_mgr Function

File

rfcomm_mgr.h

C

```
void _rfcomm_init_mgr();
```

Description

This is function _rfcomm_init_mgr.

_rfcomm_init_sessions Function

File

rfcomm_private.h

C

```
void _rfcomm_init_sessions();
```

Description

From rfcomm_session.c

_rfcomm_init_timer Function

File

rfcomm_timer.h

C

```
void _rfcomm_init_timer();
```

Description

This is function _rfcomm_init_timer.

_rfcomm_l2cap_read_data_callback Function

File

rfcomm_private.h

C

```
void _rfcomm_l2cap_read_data_callback(struct _bt_l2cap_channel_t * pch, bt_byte_p pdata, bt_int len);
```

Description

From rfcomm_cmd_recv.c

_rfcomm_mx_process_fc Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_fc(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_mx_process_fc.

_rfcomm_mx_process_pn Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_pn(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int len);
```

Description

This is function _rfcomm_mx_process_pn.

_rfcomm_mx_process_rpn Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_rpn(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int len);
```

Description

This is function _rfcomm_mx_process_rpn.

_rfcomm_process_cmd_frame_disc Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_disc(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

From frame_disc.c

_rfcomm_process_cmd_frame_dm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_dm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_dm.

_rfcomm_process_cmd_frame_sabm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_sabm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_sabm.

_rfcomm_process_cmd_frame_ua Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_ua(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_ua.

_rfcomm_process_cmd_frame_uih Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_uih(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

From frame_uih.c

_rfcomm_process_mx_fc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_fc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_mx_fc_response.

_rfcomm_process_mx_msc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_msc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_msc_response.

_rfcomm_process_mx_pn_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_pn_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pres, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_pn_response.

_rfcomm_process_mx_rls_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_rls_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_rls_response.

_rfcomm_process_res_frame_disc Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_disc(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_disc.

_rfcomm_process_res_frame_dm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_dm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_dm.

_rfcomm_process_res_frame_sabm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_sabm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_sabm.

_rfcomm_process_res_frame_ua Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_ua(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_ua.

_rfcomm_process_res_frame_uih Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_uih(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_uih.

_rfcomm_send_command Function

File

rfcomm_private.h

C

```
bt_bool _rfcomm_send_command(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

From rfcomm_cmd_send.c

_rfcomm_send_dm_response Function

File

rfcomm_private.h

C

```
void _rfcomm_send_dm_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

From frame_dm.c

_rfcomm_send_mx_fc_cmd Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_fc_cmd(bt_rfcomm_dlc_p pdlc, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_send_mx_fc_cmd.

_rfcomm_send_mx_msc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_msc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int
len);
```

Description

This is function _rfcomm_send_mx_msc_response.

_rfcomm_send_mx_nsc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_nsc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_send_mx_nsc_response.

_rfcomm_send_mx_rls_cmd Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_rls_cmd(bt_rfcomm_dlc_p pdlc);
```

Description

This is function _rfcomm_send_mx_rls_cmd.

_rfcomm_send_mx_rls_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_rls_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int
len);
```

Description

This is function _rfcomm_send_mx_rls_response.

_rfcomm_send_mx_test_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_test_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_send_mx_test_response.

_rfcomm_send_sabm_cmd Function

File

rfcomm_private.h

C

```
void _rfcomm_send_sabm_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

From frame_sabm.c

_rfcomm_send_ua_response Function

File

rfcomm_private.h

C

```
void _rfcomm_send_ua_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_rfcomm_cmd_callback_fp cb);
```

Description

From frame_ua.c

_bt_rfcomm_allocate_channel Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_server_channel_t* _bt_rfcomm_allocate_channel(bt_byte id);
```

Description

This is function _bt_rfcomm_allocate_channel.

_bt_rfcomm_clear_queue Function

File

rfcomm_private.h

C

```
void _bt_rfcomm_clear_queue(bt_rfcomm_dlc_t* pdlc);
```

Description

From command_queue.c

_bt_rfcomm_find_channel Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_server_channel_t* _bt_rfcomm_find_channel(bt_byte id);
```

Description

This is function _bt_rfcomm_find_channel.

_bt_rfcomm_get_mgr Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_mgr_t* _bt_rfcomm_get_mgr();
```

Description

This is function _bt_rfcomm_get_mgr.

_bt_rfcomm_init_signal Function

File

rfcomm_signal.h

C

```
void _bt_rfcomm_init_signal();
```

Description

This is function _bt_rfcomm_init_signal.

_bt_rfcomm_mgr_allocate_session Function

File

rfcomm_private.h

C

```
bt_rfcomm_session_t* _bt_rfcomm_mgr_allocate_session(bt_bdaddr_t* bdaddr_remote);
```

Description

This is function _bt_rfcomm_mgr_allocate_session.

_bt_rfcomm_mgr_l2cap_listen_callback Function

File

rfcomm_mgr.h

C

```
void _bt_rfcomm_mgr_l2cap_listen_callback(bt_l2cap_channel_t* pch, void* param);
```

Description

This is function _bt_rfcomm_mgr_l2cap_listen_callback.

_bt_rfcomm_mgr_notify_listeners Function

File

rfcomm_mgr.h

C

```
void _bt_rfcomm_mgr_notify_listeners(bt_rfcomm_session_t* session, bt_int evt);
```

Description

This is function _bt_rfcomm_mgr_notify_listeners.

_bt_rfcomm_set_signal Function

File

rfcomm_signal.h

C

```
void _bt_rfcomm_set_signal();
```

Description

This is function _bt_rfcomm_set_signal.

_calc_fcs Function

File

rfcomm_private.h

C

```
bt_byte _calc_fcs(bt_byte_p buffer, bt_byte len);
```

Description

This is function _calc_fcs.

rfcomm_find_session Function

File

rfcomm_private.h

C

```
bt_rfcomm_session_p rfcomm_find_session(bt_l2cap_channel_t* pch);
```

Description

This is function rfcomm_find_session.

_rfcomm_l2cap_state_changed_callback Function

File

rfcomm_private.h

C

```
void _rfcomm_l2cap_state_changed_callback(bt_l2cap_channel_t* pch, bt_int new_state, void* param);
```

Description

From rfcomm.c

bt_rfcomm_cancel_listen Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_cancel_listen(bt_byte server_channel);
```

Description

This is function bt_rfcomm_cancel_listen.

bt_rfcomm_get_frame_length Function

File

rfcomm.h

C

```
bt_int bt_rfcomm_get_frame_length(bt_rfcomm_dlc_t* dlc);
```

Description

This is function bt_rfcomm_get_frame_length.

rfcomm_cq_find_failed_pn Function

File

rfcomm_cmd_queue.h

C

```
struct _bt_rfcomm_command_t* rfcomm_cq_find_failed_pn(struct _bt_rfcomm_dlc_t* pdlc, struct
_bt_rfcomm_command_t* dm);
```

Description

This is function rfcomm_cq_find_failed_pn.

RFCOMM Data Types and Constants

MK_CMD_ADDRESS Macro

File

rfcomm.h

C

```
#define MK_CMD_ADDRESS(addr, pdlc) ((addr & 0xfc) | pdlc->psess->role | RFCOMM_FLAG_EA)
```

Description

define RFCOMM_MS_RTC

MK_DLCI Macro

File

rfcomm.h

C

```
#define MK_DLCI(bt_server_channel, rfcomm_session) (bt_server_channel << 3 | ((~rfcomm_session->role & 2)
<< 1))
```

Description

This is macro MK_DLCI.

RFCOMM_CFC_ENABLED Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_ENABLED (pdlc->psess->fc_type == RFCOMM_FC_TYPE_CREDIT)
```

Description

This is macro RFCOMM_CFC_ENABLED.

RFCOMM_CFC_LOCAL_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_LOCAL_CREDIT _rfcomm_local_credit
```

Description

This is macro RFCOMM_CFC_LOCAL_CREDIT.

RFCOMM_CFC_MAX_INITIAL_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_MAX_INITIAL_CREDIT 7
```

Description

This is macro RFCOMM_CFC_MAX_INITIAL_CREDIT.

RFCOMM_CMD_STATUS_FC_PENDING Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_FC_PENDING 2
```

Description

This is macro RFCOMM_CMD_STATUS_FC_PENDING.

RFCOMM_CMD_STATUS_PENDING Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_PENDING 0
```

Description

This is macro RFCOMM_CMD_STATUS_PENDING.

RFCOMM_CMD_STATUS_WAITING_RESPONSE Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro RFCOMM_CMD_STATUS_WAITING_RESPONSE.

RFCOMM_COMMAND Macro

File

rfcomm.h

C

```
#define RFCOMM_COMMAND 0x02
```

Description

This is macro RFCOMM_COMMAND.

RFCOMM_CTL_MSG_CLD Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_CLD 0xc0
```

Description

This is macro RFCOMM_CTL_MSG_CLD.

RFCOMM_CTL_MSG_FCOFF Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_FCOFF 0x60
```

Description

This is macro RFCOMM_CTL_MSG_FCOFF.

RFCOMM_CTL_MSG_FCON Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_FCON 0x10
```

Description

This is macro RFCOMM_CTL_MSG_FCON.

RFCOMM_CTL_MSG_MSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_MSC 0xe0
```

Description

This is macro RFCOMM_CTL_MSG_MSC.

RFCOMM_CTL_MSG_NSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_NSC 0x10
```

Description

This is macro RFCOMM_CTL_MSG_NSC.

RFCOMM_CTL_MSG_PN Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_PN 0x80
```

Description

This is macro RFCOMM_CTL_MSG_PN.

RFCOMM_CTL_MSG_PSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_PSC 0x40
```

Description

This is macro RFCOMM_CTL_MSG_PSC.

RFCOMM_CTL_MSG_RLS Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_RLS 0x50
```

Description

This is macro RFCOMM_CTL_MSG_RLS.

RFCOMM_CTL_MSG_RPN Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_RPN 0x90
```

Description

This is macro RFCOMM_CTL_MSG_RPN.

RFCOMM_CTL_MSG_SNC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_SNC 0xd0
```

Description

This is macro RFCOMM_CTL_MSG_SNC.

RFCOMM_CTL_MSG_TEST Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_TEST 0x20
```

Description

This is macro RFCOMM_CTL_MSG_TEST.

RFCOMM_DLC_CHANGED_CONN_STATE Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CHANGED_CONN_STATE 0
```

Description

This is macro RFCOMM_DLC_CHANGED_CONN_STATE.

RFCOMM_DLC_CHANGED_REMOTE_MSC Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CHANGED_REMOTE_MSC 1
```

Description

This is macro RFCOMM_DLC_CHANGED_REMOTE_MSC.

RFCOMM_DLC_CONNECTION_FAILED Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CONNECTION_FAILED 2
```

Description

This is macro RFCOMM_DLC_CONNECTION_FAILED.

RFCOMM_DLC_STATE_CLOSED Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_STATE_CLOSED 0
```

Description

This is macro RFCOMM_DLC_STATE_CLOSED.

RFCOMM_DLC_STATE_OPEN Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_STATE_OPEN 1
```

Description

This is macro RFCOMM_DLC_STATE_OPEN.

RFCOMM_DLCI_CONTROL Macro

File

rfcomm.h

C

```
#define RFCOMM_DLCI_CONTROL 0
```

Description

This is macro RFCOMM_DLCI_CONTROL.

RFCOMM_DLCI_FREE Macro

File

rfcomm.h

C

```
#define RFCOMM_DLCI_FREE 0xff
```

Description

This is macro RFCOMM_DLCI_FREE.

RFCOMM_ERR_DM Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_DM 0x02
```

Description

This is macro RFCOMM_ERR_DM.

RFCOMM_ERR_SUCCESS Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_SUCCESS 0x00
```

Description

This is macro RFCOMM_ERR_SUCCESS.

RFCOMM_ERR_TIMEOUT Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_TIMEOUT 0x01
```

Description

This is macro RFCOMM_ERR_TIMEOUT.

RFCOMM_FC_TYPE_AGREGATE Macro

File

rfcomm.h

C

```
#define RFCOMM_FC_TYPE_AGREGATE 0
```

Description

This is macro RFCOMM_FC_TYPE_AGREGATE.

RFCOMM_FC_TYPE_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_FC_TYPE_CREDIT 1
```

Description

This is macro RFCOMM_FC_TYPE_CREDIT.

RFCOMM_FLAG_CR Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_CR 0x02
```

Description

This is macro RFCOMM_FLAG_CR.

RFCOMM_FLAG_EA Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_EA 0x01
```

Description

This is macro RFCOMM_FLAG_EA.

RFCOMM_FLAG_PF Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_PF 0x10
```

Description

This is macro RFCOMM_FLAG_PF.

RFCOMM_FRAME_HEADER_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_HEADER_LEN 5
```

Description

This is macro RFCOMM_FRAME_HEADER_LEN.

RFCOMM_FRAME_TYPE_DISC Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_DISC 0x43
```

Description

This is macro RFCOMM_FRAME_TYPE_DISC.

RFCOMM_FRAME_TYPE_DM Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_DM 0x0f
```

Description

This is macro RFCOMM_FRAME_TYPE_DM.

RFCOMM_FRAME_TYPE_SABM Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_SABM 0x2f
```

Description

This is macro RFCOMM_FRAME_TYPE_SABM.

RFCOMM_FRAME_TYPE_UA Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UA 0x63
```

Description

This is macro RFCOMM_FRAME_TYPE_UA.

RFCOMM_FRAME_TYPE_UI Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UI 0x03
```

Description

This is macro RFCOMM_FRAME_TYPE_UI.

RFCOMM_FRAME_TYPE_UIH Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UIH 0xef
```

Description

This is macro RFCOMM_FRAME_TYPE_UIH.

RFCOMM_LINE_STATUS_FRAMING Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_FRAMING x09
```

Description

This is macro RFCOMM_LINE_STATUS_FRAMING.

RFCOMM_LINE_STATUS_OVERRUN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_OVERRUN x03
```

Description

This is macro RFCOMM_LINE_STATUS_OVERRUN.

RFCOMM_LINE_STATUS_PARITY Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_PARITY x05
```

Description

This is macro RFCOMM_LINE_STATUS_PARITY.

RFCOMM_MAX_INFO_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MAX_INFO_LEN _rfcomm_info_len
```

Description

This is macro RFCOMM_MAX_INFO_LEN.

RFCOMM_MODEM_STATUS_DV Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_DV 0x80
```

Description

This is macro RFCOMM_MODEM_STATUS_DV.

RFCOMM_MODEM_STATUS_FC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_FC 0x02
```

Description

This is macro RFCOMM_MODEM_STATUS_FC.

RFCOMM_MODEM_STATUS_IC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_IC 0x40
```

Description

This is macro RFCOMM_MODEM_STATUS_IC.

RFCOMM_MODEM_STATUS_RTC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_RTC 0x04
```

Description

This is macro RFCOMM_MODEM_STATUS_RTC.

RFCOMM_MODEM_STATUS_RTR Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_RTR 0x08
```

Description

This is macro RFCOMM_MODEM_STATUS_RTR.

RFCOMM_MX_MSG_FCOFF Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_FCOFF 0x60
```

Description

This is macro RFCOMM_MX_MSG_FCOFF.

RFCOMM_MX_MSG_FCON Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_FCON 0xA0
```

Description

This is macro RFCOMM_MX_MSG_FCON.

RFCOMM_MX_MSG_MSC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_MSC 0xE0
```

Description

This is macro RFCOMM_MX_MSG_MSC.

RFCOMM_MX_MSG_NSC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_NSC 0x10
```

Description

This is macro RFCOMM_MX_MSG_NSC.

RFCOMM_MX_MSG_PN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_PN 0x80
```

Description

This is macro RFCOMM_MX_MSG_PN.

RFCOMM_MX_MSG_RLS Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_RLS 0x50
```

Description

This is macro RFCOMM_MX_MSG_RLS.

RFCOMM_MX_MSG_RPN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_RPN 0x90
```

Description

This is macro RFCOMM_MX_MSG_RPN.

RFCOMM_MX_MSG_TEST Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_TEST 0x20
```

Description

This is macro RFCOMM_MX_MSG_TEST.

RFCOMM_RESPONSE Macro

File

rfcomm.h

C

```
#define RFCOMM_RESPONSE 0x00
```

Description

This is macro RFCOMM_RESPONSE.

RFCOMM_ROLE_INITIATOR Macro

File

rfcomm.h

C

```
#define RFCOMM_ROLE_INITIATOR 0x02
```

Description

This is macro RFCOMM_ROLE_INITIATOR.

RFCOMM_ROLE_RESPONDER Macro

File

rfcomm.h

C

```
#define RFCOMM_ROLE_RESPONDER 0x00
```

Description

This is macro RFCOMM_ROLE_RESPONDER.

RFCOMM_RPN_BAUD_RATE_1152 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_1152 0x07
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_1152.

RFCOMM_RPN_BAUD_RATE_192 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_192 0x04
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_192.

RFCOMM_RPN_BAUD_RATE_2304 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_2304 0x08
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_2304.

RFCOMM_RPN_BAUD_RATE_24 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_24 0x00
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_24.

RFCOMM_RPN_BAUD_RATE_384 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_384 0x05
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_384.

RFCOMM_RPN_BAUD_RATE_48 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_48 0x01
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_48.

RFCOMM_RPN_BAUD_RATE_576 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_576 0x06
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_576.

RFCOMM_RPN_BAUD_RATE_72 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_72 0x02
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_72.

RFCOMM_RPN_BAUD_RATE_96 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_96 0x03 // default
```

Description

default

RFCOMM_RPN_DATA_BIT_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_5 0x00
```

Description

This is macro RFCOMM_RPN_DATA_BIT_5.

RFCOMM_RPN_DATA_BIT_6 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_6 0x01
```

Description

This is macro RFCOMM_RPN_DATA_BIT_6.

RFCOMM_RPN_DATA_BIT_7 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_7 0x02
```

Description

This is macro RFCOMM_RPN_DATA_BIT_7.

RFCOMM_RPN_DATA_BIT_8 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_8 0x03 // default
```

Description

default

RFCOMM_RPN_FLC_N Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_N 0x00
```

Description

This is macro RFCOMM_RPN_FLC_N.

RFCOMM_RPN_FLC_RTC_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTC_INPUT 0x10
```

Description

This is macro RFCOMM_RPN_FLC_RTC_INPUT.

RFCOMM_RPN_FLC_RTC_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTC_OUTPUT 0x20
```

Description

This is macro RFCOMM_RPN_FLC_RTC_OUTPUT.

RFCOMM_RPN_FLC_RTR_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTR_INPUT 0x04
```

Description

This is macro RFCOMM_RPN_FLC_RTR_INPUT.

RFCOMM_RPN_FLC_RTR_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTR_OUTPUT 0x08
```

Description

This is macro RFCOMM_RPN_FLC_RTR_OUTPUT.

RFCOMM_RPN_FLC_XONOFF_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_XONOFF_INPUT 0x01
```

Description

This is macro RFCOMM_RPN_FLC_XONOFF_INPUT.

RFCOMM_RPN_FLC_XONOFF_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_XONOFF_OUTPUT 0x02
```

Description

This is macro RFCOMM_RPN_FLC_XONOFF_OUTPUT.

RFCOMM_RPN_PARITY_EVEN Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_EVEN 0x10
```

Description

This is macro RFCOMM_RPN_PARITY_EVEN.

RFCOMM_RPN_PARITY_MARK Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_MARK 0x20
```

Description

This is macro RFCOMM_RPN_PARITY_MARK.

RFCOMM_RPN_PARITY_N Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_N 0x00 // default
```

Description

default

RFCOMM_RPN_PARITY_ODD Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_ODD 0x00
```

Description

This is macro RFCOMM_RPN_PARITY_ODD.

RFCOMM_RPN_PARITY_SPACE Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_SPACE 0x30
```

Description

This is macro RFCOMM_RPN_PARITY_SPACE.

RFCOMM_RPN_PARITY_Y Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_Y 0x08
```

Description

This is macro RFCOMM_RPN_PARITY_Y.

RFCOMM_RPN_STOP_BIT_1 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_STOP_BIT_1 0x00 // default
```

Description

default

RFCOMM_RPN_STOP_BIT_1_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_STOP_BIT_1_5 0x04
```

Description

This is macro RFCOMM_RPN_STOP_BIT_1_5.

RFCOMM_RPN_XOFF_DEFAULT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_XOFF_DEFAULT 0x13
```

Description

This is macro RFCOMM_RPN_XOFF_DEFAULT.

RFCOMM_RPN_XON_DEFAULT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_XON_DEFAULT 0x11
```

Description

This is macro RFCOMM_RPN_XON_DEFAULT.

RFCOMM_SERIAL_PORT_CH_1 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_1 1
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_1.

RFCOMM_SERIAL_PORT_CH_10 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_10 10
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_10.

RFCOMM_SERIAL_PORT_CH_11 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_11 11
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_11.

RFCOMM_SERIAL_PORT_CH_12 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_12 12
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_12.

RFCOMM_SERIAL_PORT_CH_13 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_13 13
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_13.

RFCOMM_SERIAL_PORT_CH_14 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_14 14
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_14.

RFCOMM_SERIAL_PORT_CH_15 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_15 15
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_15.

RFCOMM_SERIAL_PORT_CH_16 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_16 16
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_16.

RFCOMM_SERIAL_PORT_CH_17 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_17 17
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_17.

RFCOMM_SERIAL_PORT_CH_18 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_18 18
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_18.

RFCOMM_SERIAL_PORT_CH_19 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_19 19
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_19.

RFCOMM_SERIAL_PORT_CH_2 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_2 2
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_2.

RFCOMM_SERIAL_PORT_CH_20 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_20 20
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_20.

RFCOMM_SERIAL_PORT_CH_21 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_21 21
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_21.

RFCOMM_SERIAL_PORT_CH_22 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_22 22
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_22.

RFCOMM_SERIAL_PORT_CH_23 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_23 23
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_23.

RFCOMM_SERIAL_PORT_CH_24 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_24 24
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_24.

RFCOMM_SERIAL_PORT_CH_25 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_25 25
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_25.

RFCOMM_SERIAL_PORT_CH_26 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_26 26
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_26.

RFCOMM_SERIAL_PORT_CH_27 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_27 27
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_27.

RFCOMM_SERIAL_PORT_CH_28 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_28 28
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_28.

RFCOMM_SERIAL_PORT_CH_29 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_29 29
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_29.

RFCOMM_SERIAL_PORT_CH_3 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_3 3
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_3.

RFCOMM_SERIAL_PORT_CH_30 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_30 30
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_30.

RFCOMM_SERIAL_PORT_CH_4 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_4 4
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_4.

RFCOMM_SERIAL_PORT_CH_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_5 5
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_5.

RFCOMM_SERIAL_PORT_CH_6 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_6 6
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_6.

RFCOMM_SERIAL_PORT_CH_7 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_7 7
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_7.

RFCOMM_SERIAL_PORT_CH_8 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_8 8
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_8.

RFCOMM_SERIAL_PORT_CH_9 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_9 9
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_9.

RFCOMM_SESSION_CHANGED_AFC Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_CHANGED_AFC 1
```

Description

This is macro RFCOMM_SESSION_CHANGED_AFC.

RFCOMM_SESSION_CHANGED_CONN_STATE Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_CHANGED_CONN_STATE 0
```

Description

This is macro RFCOMM_SESSION_CHANGED_CONN_STATE.

RFCOMM_SESSION_STATE_CONNECTED Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_CONNECTED 1
```

Description

This is macro RFCOMM_SESSION_STATE_CONNECTED.

RFCOMM_SESSION_STATE_DISCONNECTED Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_DISCONNECTED 2
```

Description

This is macro RFCOMM_SESSION_STATE_DISCONNECTED.

RFCOMM_SESSION_STATE_FREE Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_FREE 0
```

Description

This is macro RFCOMM_SESSION_STATE_FREE.

RFCOMM_TIMEOUT Macro

File

rfcomm.h

C

```
#define RFCOMM_TIMEOUT 60
```

Description

This is macro RFCOMM_TIMEOUT.

bt_rfcomm_cmd_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_cmd_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, struct _bt_rfcomm_command_t *cmd,
bt_int status);
```

Description

This is type bt_rfcomm_cmd_callback_fp.

bt_rfcomm_command_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_command_t {
  struct _bt_rfcomm_command_t* next_cmd;
  bt_byte address;
  bt_byte control;
  bt_byte credit;
  bt_byte_p pdata;
  bt_int len;
  bt_byte status;
  bt_rfcomm_cmd_callback_fp cb;
  bt_rfcomm_send_data_callback_fp send_data_cb;
  bt_long send_time;
  bt_byte mx_params[RFCOMM_MX_MSG_MAX_DATA_LEN];
} bt_rfcomm_command_t, * bt_rfcomm_command_p;
```

Description

This is type bt_rfcomm_command_p.

bt_rfcomm_command_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_command_t {
  struct _bt_rfcomm_command_t* next_cmd;
  bt_byte address;
  bt_byte control;
  bt_byte credit;
  bt_byte_p pdata;
  bt_int len;
  bt_byte status;
  bt_rfcomm_cmd_callback_fp cb;
  bt_rfcomm_send_data_callback_fp send_data_cb;
```

```
    bt_long send_time;
    bt_byte mx_params[RFCOMM_MX_MSG_MAX_DATA_LEN];
  } bt_rfcomm_command_t, * bt_rfcomm_command_p;
```

Description

This is type bt_rfcomm_command_t.

bt_rfcomm_ctl_msg_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_ctl_msg_t {
  bt_byte type;
  bt_int len;
  bt_byte_p pdata;
} bt_rfcomm_ctl_msg_t, * bt_rfcomm_ctl_msg_p;
```

Description

This is type bt_rfcomm_ctl_msg_p.

bt_rfcomm_ctl_msg_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_ctl_msg_t {
  bt_byte type;
  bt_int len;
  bt_byte_p pdata;
} bt_rfcomm_ctl_msg_t, * bt_rfcomm_ctl_msg_p;
```

Description

This is type bt_rfcomm_ctl_msg_t.

bt_rfcomm_dlc_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_dlc_t {
  bt_byte id;
  bt_byte priority;
  bt_uint max_frame_size;
  bt_byte cfc_remote_credit;
  bt_byte cfc_freed_local_buffers;
  bt_byte state;
  bt_byte local_ms;
  bt_byte remote_ms;
  bt_byte local_ls;
  bt_byte remote_ls;
  bt_bool buffered;
  bt_rfcomm_read_data_callback_fp read_data_cb;
  bt_rfcomm_dlc_state_callback_fp state_cb;
  void* cb_param;
  bt_rfcomm_packet_t tx_packet;
  struct _bt_rfcomm_session_t * psess;
  bt_queue_element_t* cmd_queue;
} bt_rfcomm_dlc_t, * bt_rfcomm_dlc_p;
```

Description

This is type bt_rfcomm_dlc_p.

bt_rfcomm_dlc_state_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_dlc_state_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_int what, void* param);
```

Description

This is type bt_rfcomm_dlc_state_callback_fp.

bt_rfcomm_dlc_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_dlc_t {
  bt_byte id;
  bt_byte priority;
  bt_uint max_frame_size;
  bt_byte cfc_remote_credit;
  bt_byte cfc_freed_local_buffers;
  bt_byte state;
  bt_byte local_ms;
  bt_byte remote_ms;
  bt_byte local_ls;
  bt_byte remote_ls;
  bt_bool buffered;
  bt_rfcomm_read_data_callback_fp read_data_cb;
  bt_rfcomm_dlc_state_callback_fp state_cb;
  void* cb_param;
  bt_rfcomm_packet_t tx_packet;
  struct _bt_rfcomm_session_t * psess;
  bt_queue_element_t* cmd_queue;
} bt_rfcomm_dlc_t, * bt_rfcomm_dlc_p;
```

Description

This is type bt_rfcomm_dlc_t.

bt_rfcomm_mgr_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_mgr_t {
  bt_rfcomm_session_listener_t* listeners;
  bt_rfcomm_server_channel_t* channels;
} bt_rfcomm_mgr_t;
```

Description

This is type bt_rfcomm_mgr_t.

bt_rfcomm_read_data_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_read_data_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_byte_p data, bt_int len);
```

Description

This is type bt_rfcomm_read_data_callback_fp.

bt_rfcomm_send_data_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_send_data_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_byte_p data, bt_int len,
bt_int status);
```

Description

This is type bt_rfcomm_send_data_callback_fp.

bt_rfcomm_server_channel_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_server_channel_t {
  bt_byte id;
  bt_rfcomm_dlc_state_callback_fp listen_cb;
  void* listen_param;
} bt_rfcomm_server_channel_t;
```

Description

This is type bt_rfcomm_server_channel_t.

bt_rfcomm_session_listener_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_session_listener_t {
  struct _bt_rfcomm_session_listener_t* next_listener;
  bt_rfcomm_state_callback_fp callback;
  void* callback_param;
} bt_rfcomm_session_listener_t;
```

Description

This is type bt_rfcomm_session_listener_t.

bt_rfcomm_session_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_session_t {
  bt_byte state;
  bt_l2cap_channel_t* pch;
  bt_byte role;
  bt_byte fc_type;
  bt_bool afc_off;
  bt_rfcomm_dlc_t* dlcs;
  bt_rfcomm_dlc_state_callback_fp connect_cb;
  void* connect_param;
  bt_byte connect_server_channel;
  bt_rfcomm_command_p pcmd_cur;
```

```
  bt_l2cap_mgr_p l2cap_mgr;
} bt_rfcomm_session_t, * bt_rfcomm_session_p;
```

Description

This is type bt_rfcomm_session_p.

bt_rfcomm_session_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_session_t {
  bt_byte state;
  bt_l2cap_channel_t* pch;
  bt_byte role;
  bt_byte fc_type;
  bt_bool afc_off;
  bt_rfcomm_dlc_t* dlcs;
  bt_rfcomm_dlc_state_callback_fp connect_cb;
  void* connect_param;
  bt_byte connect_server_channel;
  bt_rfcomm_command_p pcmd_cur;
  bt_l2cap_mgr_p l2cap_mgr;
} bt_rfcomm_session_t, * bt_rfcomm_session_p;
```

Description

This is type bt_rfcomm_session_t.

bt_rfcomm_state_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_state_callback_fp)(struct _bt_rfcomm_session_t* session, bt_int what, void*
param);
```

Description

This is type bt_rfcomm_state_callback_fp.

RFCOMM_ERR_INTERRUPTED Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_INTERRUPTED 0x03
```

Description

This is macro RFCOMM_ERR_INTERRUPTED.

RFCOMM_MIX_INFO_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MIX_INFO_LEN 31
```

Description

This is macro RFCOMM_MIX_INFO_LEN.

RFCOMM_MX_MSG_MAX_DATA_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MX_MSG_MAX_DATA_LEN 10
```

Description

This is macro RFCOMM_MX_MSG_MAX_DATA_LEN.

SBC Decoder Functions

sbc_decode Function

Decodes the encoded SBC frame.

File

sbc.h

C

```
ssize_t sbc_decode(sbc_t * sbc, const void * input, size_t input_len, void * output, size_t output_len,
size_t * written);
```

Returns

This function returns the length of one encoded frame that has just been decoded.

Description

This function decodes the encoded SBC frame.

Remarks

This function is called once to decode one frame and will be called in an infinite loop. The decoder decodes one frame at a time even though there can be several frames in the input buffer. This function returns the length of the encoded frame that has just been decoded. The decoder also populates the sbc_t *sbc structure with the sampling rate, number of sub-bands, bitpool value, mode, and the allocation method.

Parameters

Parameters	Description
*sbc	Pointer to the SBC data structure.
*input	The pointer to the input buffer, where the encoded bit stream is available.
input_len	The number of valid input data bytes available in the above buffer.
*output	The pointer to the output sample buffer where the decoded PCM samples are to be written.
output_len	Maximum length of the output buffer.
*written	The number of output bytes written in the above buffer.

Function

ssize_t sbc_decode(sbc_t *sbc, const void *input, size_t input_len,

void *output, size_t output_len, size_t *written);

sbc_get_codesize Function

Gets the input block size.

File

sbc.h

C

```
size_t sbc_get_codesize(sbc_t * sbc);
```

Returns

This function returns the size of the encoded frame in bytes including the 4-byte header.

Description

This function gets the input block size.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

size_t sbc_get_codesize(sbc_t *sbc);

sbc_get_frame_duration Function

Gets the time one input/output block takes to play in microseconds.

File

sbc.h

C

```
uint32_t sbc_get_frame_duration(sbc_t * sbc);
```

Returns

This function returns the duration of the last decoded frame to be played. The duration is in microseconds.

Description

This function gets the time one input/output block takes to play in microseconds.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

uint32_t sbc_get_frame_duration(sbc_t *sbc);

sbc_get_frame_length Function

Gets the output block size.

File

sbc.h

C

```
size_t sbc_get_frame_length(sbc_t * sbc);
```

Returns

This function returns the size of the output frame (in bytes) that was decoded last.

Description

This function gets the output block size.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

size_t sbc_get_frame_length(sbc_t *sbc);

sbc_get_state_size Function

Gets the SBC Decoder's state data structure.

File

sbc.h

C

```
int16_t sbc_get_state_size();
```

Returns

This function returns the size of the SBC Decoder state structure in bytes.

Description

This function gets the SBC Decoder's state data structure size.

Function

int16_t sbc_get_state_size();

sbc_init Function

Initializes the SBC Decoder.

File

sbc.h

C

```
int32_t sbc_init(sbc_t * sbc, uint32_t flags);
```

Returns

This function always returns the value '0'; however, this return value is not used in the application.

Description

This function initializes the SBC Decoder.

Remarks

This function views the received pointer as structure pointer and clears the entire structure. This function is called only once for each channel before starting the decoding of the first frame.

Parameters

Parameters	Description
*sbc	Pointer to the SBC data structure.
flags	Unused. 0L is always passed.

Function

int32_t sbc_init(sbc_t *sbc, uint32_t flags);

SBC Decoder Data Types and Constants

sbc_struct Structure

File

sbc.h

C

```
struct sbc_struct {
  uint32_t flags;
```

```
    uint8_t frequency;
    uint8_t blocks;
    uint8_t subbands;
    uint8_t mode;
    uint8_t allocation;
    uint8_t bitpool;
    uint8_t endian;
    void * priv;
    void * priv_alloc_base;
    int32_t v_vec_fix[2][256+160];
    int32_t SamplingFreq;
    uint32_t Dword;
    int16_t bitcount;
    int16_t Pos;
    int16_t codeSize;
    int16_t outSize;
    int8_t BlocksIndex;
    int8_t SubBandsIndex;
    int8_t nrof_Chnls;
    int8_t Bits;
    uint8_t CrcCheck;
    uint8_t CrcData[2];
    uint8_t join[8];
    int8_t bits[2][8];
    uint8_t * Buffer;
};
```

Members

Members	Description
int32_t v_vec_fix[2][256+160];	Below variables / buffers added

Section

Data Types

sbc_t Type

File

sbc.h

C

```
typedef struct sbc_struct sbc_t;
```

Description

This is type sbc_t.

__SBC_H Macro

File

sbc.h

C

```
#define __SBC_H
```

Description

This is macro __SBC_H.

INP_BUF_SIZE Macro

File

sbc.h

C

```
#define INP_BUF_SIZE 512     // Bytes
```

Description

Bytes

MAX_SBC_DEC_STATE_SIZE Macro

File

sbc.h

C

```
#define MAX_SBC_DEC_STATE_SIZE 3500    // Bytes
```

Description

Bytes

OUT_BUF_SIZE Macro

File

sbc.h

C

```
#define OUT_BUF_SIZE 512     // Bytes
```

Description

Bytes

SBC_AM_LOUDNESS Macro

File

sbc.h

C

```
#define SBC_AM_LOUDNESS 0x00
```

Description

Allocation method

SBC_AM_SNR Macro

File

sbc.h

C

```
#define SBC_AM_SNR 0x01
```

Description

This is macro SBC_AM_SNR.

SBC_BE Macro

File

sbc.h

C

```
#define SBC_BE 0x01
```

Description

This is macro SBC_BE.

SBC_BLK_12 Macro

File

sbc.h

C

```
#define SBC_BLK_12 0x02
```

Description

This is macro SBC_BLK_12.

SBC_BLK_16 Macro

File

sbc.h

C

```
#define SBC_BLK_16 0x03
```

Description

This is macro SBC_BLK_16.

SBC_BLK_4 Macro

File

sbc.h

C

```
#define SBC_BLK_4 0x00
```

Description

Blocks

SBC_BLK_8 Macro

File

sbc.h

C

```
#define SBC_BLK_8 0x01
```

Description

This is macro SBC_BLK_8.

SBC_FREQ_16000 Macro

File

sbc.h

C

```
#define SBC_FREQ_16000 0x00
```

Description

Sampling frequency

SBC_FREQ_32000 Macro

File

sbc.h

C

```
#define SBC_FREQ_32000 0x01
```

Description

This is macro SBC_FREQ_32000.

SBC_FREQ_44100 Macro

File

sbc.h

C

```
#define SBC_FREQ_44100 0x02
```

Description

This is macro SBC_FREQ_44100.

SBC_FREQ_48000 Macro

File

sbc.h

C

```
#define SBC_FREQ_48000 0x03
```

Description

This is macro SBC_FREQ_48000.

SBC_LE Macro

File

sbc.h

C

```
#define SBC_LE 0x00
```

Description

Data endianness

SBC_MODE_DUAL_CHANNEL Macro

File

sbc.h

C

```
#define SBC_MODE_DUAL_CHANNEL 0x01
```

Description

This is macro SBC_MODE_DUAL_CHANNEL.

SBC_MODE_JOINT_STEREO Macro

File

sbc.h

C

```
#define SBC_MODE_JOINT_STEREO 0x03
```

Description

This is macro SBC_MODE_JOINT_STEREO.

SBC_MODE_MONO Macro

File

sbc.h

C

```
#define SBC_MODE_MONO 0x00
```

Description

Channel mode

SBC_MODE_STEREO Macro

File

sbc.h

C

```
#define SBC_MODE_STEREO 0x02
```

Description

This is macro SBC_MODE_STEREO.

SBC_SB_4 Macro

File

sbc.h

C

```
#define SBC_SB_4 0x00
```

Description

Sub-bands

SBC_SB_8 Macro

File

sbc.h

C

```
#define SBC_SB_8 0x01
```

Description

This is macro SBC_SB_8.

ssize_t Macro

File

sbc.h

C

```
#define ssize_t __int32_t
```

Section

Constants

SDP Functions

bt_sdp_de_to_uuid Function

File

sdp_utils.h

C

```
void bt_sdp_de_to_uuid(bt_sdp_data_element_cp pde, bt_uuid_p puuid);
```

Description

This is function bt_sdp_de_to_uuid.

bt_sdp_packet_assembler Function

File

sdp_packet.h

C

```
bt_int bt_sdp_packet_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_sdp_packet_assembler.

bt_sdp_read_attribute Function

File

sdp_private.h

C

```
bt_bool bt_sdp_read_attribute(bt_sdp_server_attribute_t* attr, const bt_byte* buffer, bt_int len, bt_int*
offset);
```

Description

From sdp_db_utils.c

bt_sdp_start Function

File

sdp.h

C

```
bt_bool bt_sdp_start(bt_l2cap_mgr_p l2cap_mgr, const bt_byte* sdp_db, bt_uint sdp_db_len);
```

Description

brief Start SDP server ingroup sdp

details This function starts the SDP server.

param l2cap_mgr The L2CAP manager on which the SDP server is to be started. param sdp_db A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros.

return li c TRUE if the function succeeds. li c FALSE otherwise.

sdp_compare_uuid_de Function

File

sdp_private.h

C

```
bt_bool sdp_compare_uuid_de(bt_sdp_data_element_cp pde1, bt_sdp_data_element_cp pde2);
```

Description

This is function sdp_compare_uuid_de.

sdp_find_attributes Function

File

sdp_private.h

C

```
bt_bool sdp_find_attributes(bt_sr_handle_p h_list, bt_int h_count, bt_byte_p pparams, bt_int params_len,
bt_int offsetInit, bt_byte patternCount, bt_sdp_transaction_t* ptran, bt_int max_attrs);
```

Description

This is function sdp_find_attributes.

sdp_find_service_records2 Function

File

sdp_private.h

C

```
int sdp_find_service_records2(bt_byte_p pparams, bt_int params_len, bt_int offsetInit, bt_byte
patternCount, bt_sdp_service_transaction_p ptran, bt_int max_handles);
```

Description

From sdp_utils.c

_sdp_alloc_svc_tran_buffer Function

File

sdp_private.h

C

```
bt_sdp_service_transaction_p _sdp_alloc_svc_tran_buffer(bt_int id);
```

Description

This is function _sdp_alloc_svc_tran_buffer.

_sdp_alloc_tran_buffer Function

File

sdp_private.h

C

```
bt_sdp_transaction_t* _sdp_alloc_tran_buffer(bt_int id);
```

Description

This is function _sdp_alloc_tran_buffer.

_sdp_find_svc_transaction Function

File

sdp_private.h

C

```
bt_sdp_service_transaction_p _sdp_find_svc_transaction(bt_int id);
```

Description

This is function _sdp_find_svc_transaction.

_sdp_find_transaction Function

File

sdp_private.h

C

```
bt_sdp_transaction_t* _sdp_find_transaction(bt_int id);
```

Description

This is function _sdp_find_transaction.

_sdp_free_svc_tran_buffer Function

File

sdp_private.h

C

```
void _sdp_free_svc_tran_buffer(bt_sdp_service_transaction_p ptran);
```

Description

This is function _sdp_free_svc_tran_buffer.

_sdp_free_tran_buffer Function

File

sdp_private.h

C

```
void _sdp_free_tran_buffer(bt_sdp_transaction_t* ptran);
```

Description

This is function _sdp_free_tran_buffer.

_sdp_get_de_data_len Function

File

sdp_utils.h

C

```
bt_ulong _sdp_get_de_data_len(bt_sdp_data_element_cp pde);
```

Description

This is function _sdp_get_de_data_len.

_sdp_get_de_hdr_len Function

File

sdp_utils.h

C

```
bt_ulong _sdp_get_de_hdr_len(bt_sdp_data_element_cp pde);
```

Description

This is function _sdp_get_de_hdr_len.

_sdp_init_tran_buffers Function

File

sdp_private.h

C

```
bt_bool _sdp_init_tran_buffers();
```

Description

From sdp_tran_buffer.c

_sdp_read_de_header Function

File

sdp_utils.h

C

```
bt_bool _sdp_read_de_header(bt_byte_cp buffer, bt_int len, bt_int_p poffset, bt_byte_p pde_type, bt_byte_p
pde_size_index, bt_ulong_p pde_data_len);
```

Description

This is function _sdp_read_de_header.

_sdp_write_data_element Function

File

sdp_utils.h

C

```
bt_bool _sdp_write_data_element(bt_sdp_data_element_cp pde, bt_byte_p buffer, bt_int len, bt_int_p poffset);
```

Description

This is function _sdp_write_data_element.

bt_sdp_request_service_attribute Function

File

sdp_client.h

C

```
bt_bool bt_sdp_request_service_attribute(bt_l2cap_channel_t* channel, bt_sr_handle_t sr,
bt_sdp_data_element_p pattern, bt_sdp_service_attribute_callback_fp callback, void* callback_param);
```

Description

brief Search attributes ingroup sdp

details This function retrieves attribute values from a service record.

param channel The L2CAP channel used to communicate to the remote SDP server. param sr The service record handle specifies the service

record from which attribute values are to be retrieved. param pattern The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called for the first time. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter.

param callback The callback function that will be called when search has completed.

param callback_param A pointer to arbitrary data to be passed to the c callback callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_sdp_request_service_search Function

File

sdp_client.h

C

```
bt_bool bt_sdp_request_service_search(bt_l2cap_channel_t* channel, bt_sdp_data_element_p pattern,
bt_sdp_service_search_callback_fp callback, void* callback_param);
```

Description

brief Search service records ingroup sdp

details This function locates service records on a remote SDP server that match the given service search pattern.

param channel The L2CAP channel used to communicate to the remote SDP server. param pattern The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter. param callback The callback function that will be called when search has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

_bt_sdp_client_init Function

File

sdp_private.h

C

```
void _bt_sdp_client_init();
```

Description

This is function _bt_sdp_client_init.

SDP Data Types and Constants

BEGIN_DE_SEQUENCE Macro

File

sdp.h

C

```
#define BEGIN_DE_SEQUENCE(id, len) \
    static bt_sdp_data_element_t id[len];    \
    static bt_sdp_sequence_t seq_##id = { len, (bt_sdp_data_element_p)id};    \
    static void init_de_sequence_##id() {    \
        static bt_bool initialized = FALSE;    \
        if (!initialized) {    \
            bt_int i = 0;    \
            bt_int max_len = len;    \
            bt_sdp_data_element_t* cur_de = id;    \
            initialized = TRUE;
```

Description

- brief Begin a data element sequence
- ingroup sdp
- *
- details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures.
- The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute().
- For example, to find a AVRCP Target the following code can be used:

code const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL }; const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL_TARGET };

BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID) END_DE_SEQUENCE(avrcp_target_service_search)

. . .

void findAvrcpTarget(void) { INIT_DE_SEQUENCE(avrcp_target_service_search);

bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL); }

endcode

*

- param id The data element sequence identifier.
- param len The number of elements in the data element sequence.

DE_BOOL Macro

File

sdp.h

C

```
#define DE_BOOL(value) cur_de->type = SDP_DATA_TYPE_BOOL; cur_de->data.b = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a boolean data element ingroup sdp

details This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_INT Macro

File

sdp.h

C

```
#define DE_INT(value) cur_de->type = SDP_DATA_TYPE_INT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 1-byte signed integer data element ingroup sdp

details This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_STRING Macro

File

sdp.h

C

```
#define DE_STRING(value) cur_de->type = SDP_DATA_TYPE_STRING; cur_de->data.pstr = value; cur_de++;
```

Description

brief Declare a text string data element ingroup sdp

details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_STRING2 Macro

File

sdp.h

C

```
#define DE_STRING2(value, len) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.pstr = value;
cur_de->bytecount = len; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a text string data element ingroup sdp

details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value. param len The length of the data element value.

DE_UINT Macro

File

sdp.h

C

```
#define DE_UINT(value) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.b = value; cur_de++; if (++i ==
max_len) return;
```

Description

brief Declare a 1-byte unsigned integer data element ingroup sdp

details This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_UINT16 Macro

File

sdp.h

C

```
#define DE_UINT16(value) cur_de->type = SDP_DATA_TYPE_UINT16; cur_de->data.ui = value; cur_de++; if (++i ==
max_len) return;
```

Description

brief Declare a 2-byte unsigned integer data element ingroup sdp

details This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_URL Macro

File

sdp.h

C

```
#define DE_URL(value) cur_de->type = SDP_DATA_TYPE_URL; cur_de->data.purl = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a URL data element ingroup sdp

details This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value which is a pointer to a string.

DE_UUID128 Macro

File

sdp.h

C

```
#define DE_UUID128(value) cur_de->type = SDP_DATA_TYPE_UUID128; cur_de->data.uuid128 = (bt_uuid_t*)&value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 128-bit UUID data element ingroup sdp

details This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value. The value must be a name of a variable of type bt_uuid.

DE_UUID16 Macro

File

sdp.h

C

```
#define DE_UUID16(value) cur_de->type = SDP_DATA_TYPE_UUID16; cur_de->data.uuid16 = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 16-bit UUID data element ingroup sdp

details This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_UUID32 Macro

File

sdp.h

C

```
#define DE_UUID32(value) cur_de->type = SDP_DATA_TYPE_UUID32; cur_de->data.uuid32 = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 32-bit UUID data element ingroup sdp

details This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

END_DE_SEQUENCE Macro

File

sdp.h

C

```
#define END_DE_SEQUENCE(id) }}
```

Description

brief End a data element sequence ingroup sdp

details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures.

param id The data element sequence identifier.

INIT_DE_SEQUENCE Macro

File

sdp.h

C

```
#define INIT_DE_SEQUENCE(id) init_de_sequence_##id();
```

Description

brief Initialize a data element sequence ingroup sdp

details This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence.

param id The data element sequence identifier.

SDP_ATTRID_AdditionalProtocolDescriptorLists Macro

File

sdp.h

C

```
#define SDP_ATTRID_AdditionalProtocolDescriptorLists 0x000D
```

Description

This is macro SDP_ATTRID_AdditionalProtocolDescriptorLists.

SDP_ATTRID_BluetoothProfileDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_BluetoothProfileDescriptorList 0x0009
```

Description

This is macro SDP_ATTRID_BluetoothProfileDescriptorList.

SDP_ATTRID_BrowseGroupList Macro

File

sdp.h

C

```
#define SDP_ATTRID_BrowseGroupList 0x0005
```

Description

This is macro SDP_ATTRID_BrowseGroupList.

SDP_ATTRID_ClientExecutableURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_ClientExecutableURL 0x000B
```

Description

This is macro SDP_ATTRID_ClientExecutableURL.

SDP_ATTRID_DIPrimaryRecord Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIPrimaryRecord 0x204
```

Description

This is macro SDP_ATTRID_DIPrimaryRecord.

SDP_ATTRID_DIProductId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIProductId 0x202
```

Description

This is macro SDP_ATTRID_DIProductId.

SDP_ATTRID_DISpecificationId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DISpecificationId 0x200
```

Description

This is macro SDP_ATTRID_DISpecificationId.

SDP_ATTRID_DIVendorId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVendorId 0x201
```

Description

This is macro SDP_ATTRID_DIVendorId.

SDP_ATTRID_DIVendorIdSource Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVendorIdSource 0x205
```

Description

This is macro SDP_ATTRID_DIVendorIdSource.

SDP_ATTRID_DIVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVersion 0x203
```

Description

This is macro SDP_ATTRID_DIVersion.

SDP_ATTRID_DocumentationURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_DocumentationURL 0x000A
```

Description

This is macro SDP_ATTRID_DocumentationURL.

SDP_ATTRID_GAPRemoteAudioVolumeControl Macro

File

sdp.h

C

```
#define SDP_ATTRID_GAPRemoteAudioVolumeControl 0x302
```

Description

This is macro SDP_ATTRID_GAPRemoteAudioVolumeControl.

SDP_ATTRID_GroupID Macro

File

sdp.h

C

```
#define SDP_ATTRID_GroupID 0x0200
```

Description

This is macro SDP_ATTRID_GroupID.

SDP_ATTRID_HDPDataExchangeSpecification Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPDataExchangeSpecification 0x301
```

Description

This is macro SDP_ATTRID_HDPDataExchangeSpecification.

SDP_ATTRID_HDPMCAPSupportedProcedures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPMCAPSupportedProcedures 0x302
```

Description

This is macro SDP_ATTRID_HDPMCAPSupportedProcedures.

SDP_ATTRID_HDPSuportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPSuportedFeatures 0x200
```

Description

This is macro SDP_ATTRID_HDPSuportedFeatures.

SDP_ATTRID_HFPAGNetwork Macro

File

sdp.h

C

```
#define SDP_ATTRID_HFPAGNetwork 0x301
```

Description

This is macro SDP_ATTRID_HFPAGNetwork.

SDP_ATTRID_HFPSupportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HFPSupportedFeatures 0x311
```

Description

This is macro SDP_ATTRID_HFPSupportedFeatures.

SDP_ATTRID_HIDBatteryPower Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDBatteryPower 0x209
```

Description

This is macro SDP_ATTRID_HIDBatteryPower.

SDP_ATTRID_HIDBootDevice Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDBootDevice 0x20E
```

Description

This is macro SDP_ATTRID_HIDBootDevice.

SDP_ATTRID_HIDCountryCode Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDCountryCode 0x203
```

Description

This is macro SDP_ATTRID_HIDCountryCode.

SDP_ATTRID_HIDDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDescriptorList 0x206
```

Description

This is macro SDP_ATTRID_HIDDescriptorList.

SDP_ATTRID_HIDDeviceReleaseNumber Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDeviceReleaseNumber 0x200
```

Description

This is macro SDP_ATTRID_HIDDeviceReleaseNumber.

SDP_ATTRID_HIDDeviceSubclass Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDeviceSubclass 0x202
```

Description

This is macro SDP_ATTRID_HIDDeviceSubclass.

SDP_ATTRID_HIDLANGIDBaseList Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDLANGIDBaseList 0x207
```

Description

This is macro SDP_ATTRID_HIDLANGIDBaseList.

SDP_ATTRID_HIDNormallyConnectable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDNormallyConnectable 0x20D
```

Description

This is macro SDP_ATTRID_HIDNormallyConnectable.

SDP_ATTRID_HIDParserVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDParserVersion 0x201
```

Description

This is macro SDP_ATTRID_HIDParserVersion.

SDP_ATTRID_HIDProfileVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDProfileVersion 0x20B
```

Description

This is macro SDP_ATTRID_HIDProfileVersion.

SDP_ATTRID_HIDReconnectInitiate Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDReconnectInitiate 0x205
```

Description

This is macro SDP_ATTRID_HIDReconnectInitiate.

SDP_ATTRID_HIDRemoteWake Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDRemoteWake 0x20A
```

Description

This is macro SDP_ATTRID_HIDRemoteWake.

SDP_ATTRID_HIDSDPDisable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDSDPDisable 0x208
```

Description

This is macro SDP_ATTRID_HIDSDPDisable.

SDP_ATTRID_HIDSupervisionTimeout Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDSupervisionTimeout 0x20C
```

Description

This is macro SDP_ATTRID_HIDSupervisionTimeout.

SDP_ATTRID_HIDVirtualCable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDVirtualCable 0x204
```

Description

This is macro SDP_ATTRID_HIDVirtualCable.

SDP_ATTRID_IconURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_IconURL 0x000C
```

Description

This is macro SDP_ATTRID_IconURL.

SDP_ATTRID_INVALID Macro

File

sdp.h

C

```
#define SDP_ATTRID_INVALID 0xFFFF
```

Description

This is macro SDP_ATTRID_INVALID.

SDP_ATTRID_LanguageBaseAttributeIDList Macro

File

sdp.h

C

```
#define SDP_ATTRID_LanguageBaseAttributeIDList 0x0006
```

Description

This is macro SDP_ATTRID_LanguageBaseAttributeIDList.

SDP_ATTRID_OFFSET_ProviderName Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ProviderName 0x0002
```

Description

This is macro SDP_ATTRID_OFFSET_ProviderName.

SDP_ATTRID_OFFSET_ServiceDescription Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ServiceDescription 0x0001
```

Description

This is macro SDP_ATTRID_OFFSET_ServiceDescription.

SDP_ATTRID_OFFSET_ServiceName Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ServiceName 0x0000
```

Description

This is macro SDP_ATTRID_OFFSET_ServiceName.

SDP_ATTRID_PrimaryLanguageBaseId Macro

File

sdp.h

C

```
#define SDP_ATTRID_PrimaryLanguageBaseId 0x0100
```

Description

This is macro SDP_ATTRID_PrimaryLanguageBaseId.

SDP_ATTRID_ProtocolDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_ProtocolDescriptorList 0x0004
```

Description

This is macro SDP_ATTRID_ProtocolDescriptorList.

SDP_ATTRID_ServiceAvailability Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceAvailability 0x0008
```

Description

This is macro SDP_ATTRID_ServiceAvailability.

SDP_ATTRID_ServiceClassIDList Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceClassIDList 0x0001
```

Description

This is macro SDP_ATTRID_ServiceClassIDList.

SDP_ATTRID_ServiceDatabaseState Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceDatabaseState 0x0201
```

Description

This is macro SDP_ATTRID_ServiceDatabaseState.

SDP_ATTRID_ServiceID Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceID 0x0003
```

Description

This is macro SDP_ATTRID_ServiceID.

SDP_ATTRID_ServiceInfoTimeToLive Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceInfoTimeToLive 0x0007
```

Description

This is macro SDP_ATTRID_ServiceInfoTimeToLive.

SDP_ATTRID_ServiceRecordHandle Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceRecordHandle 0x0000
```

Description

This is macro SDP_ATTRID_ServiceRecordHandle.

SDP_ATTRID_ServiceRecordState Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceRecordState 0x0002
```

Description

This is macro SDP_ATTRID_ServiceRecordState.

SDP_ATTRID_SupportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_SupportedFeatures 0x311
```

Description

This is macro SDP_ATTRID_SupportedFeatures.

SDP_ATTRID_VersionNumberList Macro

File

sdp.h

C

```
#define SDP_ATTRID_VersionNumberList 0x0200
```

Description

This is macro SDP_ATTRID_VersionNumberList.

SDP_CLIENT_EVT_CONNECTED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_CONNECTED 1
```

Description

This is macro SDP_CLIENT_EVT_CONNECTED.

SDP_CLIENT_EVT_DISCONNECTED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_DISCONNECTED 2
```

Description

This is macro SDP_CLIENT_EVT_DISCONNECTED.

SDP_CLIENT_EVT_NULL Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_NULL 0
```

Description

This is macro SDP_CLIENT_EVT_NULL.

SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION Macro

File

sdp.h

C

```
#define SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION 0x110D
```

Description

This is macro SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION.

SDP_CLSID_AUDIO_SINK Macro

File

sdp.h

C

```
#define SDP_CLSID_AUDIO_SINK 0x110B
```

Description

This is macro SDP_CLSID_AUDIO_SINK.

SDP_CLSID_AUDIO_SOURCE Macro

File

sdp.h

C

```
#define SDP_CLSID_AUDIO_SOURCE 0x110A
```

Description

This is macro SDP_CLSID_AUDIO_SOURCE.

SDP_CLSID_AV_REMOTE_CONTROL Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL 0x110E
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL.

SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER 0x110F
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER.

SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID 0x110E
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID.

SDP_CLSID_AV_REMOTE_CONTROL_TARGET Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_TARGET 0x110C
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_TARGET.

SDP_CLSID_AVCTP Macro

File

sdp.h

C

```
#define SDP_CLSID_AVCTP 0x17
```

Description

This is macro SDP_CLSID_AVCTP.

SDP_CLSID_AVDTP Macro

File

sdp.h

C

```
#define SDP_CLSID_AVDTP 0x19
```

Description

This is macro SDP_CLSID_AVDTP.

SDP_CLSID_BrowseGroupDescriptorServiceClassID Macro

File

sdp.h

C

```
#define SDP_CLSID_BrowseGroupDescriptorServiceClassID 0x1001
```

Description

This is macro SDP_CLSID_BrowseGroupDescriptorServiceClassID.

SDP_CLSID_DialupNetworking Macro

File

sdp.h

C

```
#define SDP_CLSID_DialupNetworking 0x1103
```

Description

This is macro SDP_CLSID_DialupNetworking.

SDP_CLSID_GENERIC_AUDIO Macro

File

sdp.h

C

```
#define SDP_CLSID_GENERIC_AUDIO 0x1203
```

Description

This is macro SDP_CLSID_GENERIC_AUDIO.

SDP_CLSID_HDP Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP 0x1400
```

Description

This is macro SDP_CLSID_HDP.

SDP_CLSID_HDP_SINK Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP_SINK 0x1402
```

Description

This is macro SDP_CLSID_HDP_SINK.

SDP_CLSID_HDP_SOURCE Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP_SOURCE 0x1401
```

Description

This is macro SDP_CLSID_HDP_SOURCE.

SDP_CLSID_HFP Macro

File

sdp.h

C

```
#define SDP_CLSID_HFP 0x111E
```

Description

This is macro SDP_CLSID_HFP.

SDP_CLSID_HFP_AG Macro

File

sdp.h

C

```
#define SDP_CLSID_HFP_AG 0x111F
```

Description

This is macro SDP_CLSID_HFP_AG.

SDP_CLSID_HID Macro

File

sdp.h

C

```
#define SDP_CLSID_HID 0x1124
```

Description

This is macro SDP_CLSID_HID.

SDP_CLSID_HIDProtocol Macro

File

sdp.h

C

```
#define SDP_CLSID_HIDProtocol 0x11
```

Description

This is macro SDP_CLSID_HIDProtocol.

SDP_CLSID_HSP Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP 0x1108
```

Description

This is macro SDP_CLSID_HSP.

SDP_CLSID_HSP_AG Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP_AG 0x1112
```

Description

This is macro SDP_CLSID_HSP_AG.

SDP_CLSID_HSP_HS Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP_HS 0x1131
```

Description

This is macro SDP_CLSID_HSP_HS.

SDP_CLSID_L2CAP Macro

File

sdp.h

C

```
#define SDP_CLSID_L2CAP 0x100
```

Description

This is macro SDP_CLSID_L2CAP.

SDP_CLSID_MCAP_CONTROL Macro

File

sdp.h

C

```
#define SDP_CLSID_MCAP_CONTROL 0x001E
```

Description

This is macro SDP_CLSID_MCAP_CONTROL.

SDP_CLSID_MCAP_DATA Macro

File

sdp.h

C

```
#define SDP_CLSID_MCAP_DATA 0x001F
```

Description

This is macro SDP_CLSID_MCAP_DATA.

SDP_CLSID_OBEXFileTransfer Macro

File

sdp.h

C

```
#define SDP_CLSID_OBEXFileTransfer 0x1106
```

Description

This is macro SDP_CLSID_OBEXFileTransfer.

SDP_CLSID_OBEXObjectPush Macro

File

sdp.h

C

```
#define SDP_CLSID_OBEXObjectPush 0x1105
```

Description

This is macro SDP_CLSID_OBEXObjectPush.

SDP_CLSID_PBAP_PCE Macro

File

sdp.h

C

```
#define SDP_CLSID_PBAP_PCE 0x112E
```

Description

This is macro SDP_CLSID_PBAP_PCE.

SDP_CLSID_PBAP_PSE Macro

File

sdp.h

C

```
#define SDP_CLSID_PBAP_PSE 0x112F
```

Description

This is macro SDP_CLSID_PBAP_PSE.

SDP_CLSID_PNPInformation Macro

File

sdp.h

C

```
#define SDP_CLSID_PNPInformation 0x1200
```

Description

This is macro SDP_CLSID_PNPInformation.

SDP_CLSID_PublicBrowseGroup Macro

File

sdp.h

C

```
#define SDP_CLSID_PublicBrowseGroup 0x1002
```

Description

This is macro SDP_CLSID_PublicBrowseGroup.

SDP_CLSID_RFCOMM Macro

File

sdp.h

C

```
#define SDP_CLSID_RFCOMM 0x3
```

Description

This is macro SDP_CLSID_RFCOMM.

SDP_CLSID_SerialPort Macro

File

sdp.h

C

```
#define SDP_CLSID_SerialPort 0x1101
```

Description

This is macro SDP_CLSID_SerialPort.

SDP_CLSID_ServiceDiscoveryServerServiceClassID Macro

File

sdp.h

C

```
#define SDP_CLSID_ServiceDiscoveryServerServiceClassID 0x1000
```

Description

defgroup sdp SDP

This module describe functions and data structures used to start the SDP server and perform SDP queries.

SDP_DATA_TYPE_ALTERNATIVE Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_ALTERNATIVE 7
```

Description

This is macro SDP_DATA_TYPE_ALTERNATIVE.

SDP_DATA_TYPE_BOOL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_BOOL 5
```

Description

This is macro SDP_DATA_TYPE_BOOL.

SDP_DATA_TYPE_INT Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT 2
```

Description

This is macro SDP_DATA_TYPE_INT.

SDP_DATA_TYPE_INT128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT128 0x82
```

Description

This is macro SDP_DATA_TYPE_INT128.

SDP_DATA_TYPE_INT16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT16 0x12
```

Description

This is macro SDP_DATA_TYPE_INT16.

SDP_DATA_TYPE_INT32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT32 0x22
```

Description

This is macro SDP_DATA_TYPE_INT32.

SDP_DATA_TYPE_INT64 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT64 0x42
```

Description

This is macro SDP_DATA_TYPE_INT64.

SDP_DATA_TYPE_INT8 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT8 SDP_DATA_TYPE_INT
```

Description

This is macro SDP_DATA_TYPE_INT8.

SDP_DATA_TYPE_NIL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_NIL 0
```

Description

This is macro SDP_DATA_TYPE_NIL.

SDP_DATA_TYPE_SEQUENCE Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_SEQUENCE 6
```

Description

This is macro SDP_DATA_TYPE_SEQUENCE.

SDP_DATA_TYPE_STRING Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_STRING 4
```

Description

This is macro SDP_DATA_TYPE_STRING.

SDP_DATA_TYPE_UINT Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT 1
```

Description

This is macro SDP_DATA_TYPE_UINT.

SDP_DATA_TYPE_UINT128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT128 0x81
```

Description

This is macro SDP_DATA_TYPE_UINT128.

SDP_DATA_TYPE_UINT16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT16 0x11
```

Description

This is macro SDP_DATA_TYPE_UINT16.

SDP_DATA_TYPE_UINT32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT32 0x21
```

Description

This is macro SDP_DATA_TYPE_UINT32.

SDP_DATA_TYPE_UINT64 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT64 0x41
```

Description

This is macro SDP_DATA_TYPE_UINT64.

SDP_DATA_TYPE_UINT8 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT8 SDP_DATA_TYPE_UINT
```

Description

This is macro SDP_DATA_TYPE_UINT8.

SDP_DATA_TYPE_URL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_URL 8
```

Description

This is macro SDP_DATA_TYPE_URL.

SDP_DATA_TYPE_UUID Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID 3
```

Description

This is macro SDP_DATA_TYPE_UUID.

SDP_DATA_TYPE_UUID128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID128 0x23
```

Description

This is macro SDP_DATA_TYPE_UUID128.

SDP_DATA_TYPE_UUID16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID16 SDP_DATA_TYPE_UUID
```

Description

This is macro SDP_DATA_TYPE_UUID16.

SDP_DATA_TYPE_UUID32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID32 0x13
```

Description

This is macro SDP_DATA_TYPE_UUID32.

SDP_ERROR_INSUFFICIENT_RESOURCE Macro

File

sdp.h

C

```
#define SDP_ERROR_INSUFFICIENT_RESOURCE 0x06
```

Description

This is macro SDP_ERROR_INSUFFICIENT_RESOURCE.

SDP_ERROR_INVALID_CONTINUATION_STATE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_CONTINUATION_STATE 0x05
```

Description

This is macro SDP_ERROR_INVALID_CONTINUATION_STATE.

SDP_ERROR_INVALID_PDU_SIZE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_PDU_SIZE 0x04
```

Description

This is macro SDP_ERROR_INVALID_PDU_SIZE.

SDP_ERROR_INVALID_REQUEST_SYNTAX Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_REQUEST_SYNTAX 0x03
```

Description

This is macro SDP_ERROR_INVALID_REQUEST_SYNTAX.

SDP_ERROR_INVALID_SDP_VERSION Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_SDP_VERSION 0x01
```

Description

This is macro SDP_ERROR_INVALID_SDP_VERSION.

SDP_ERROR_INVALID_SR_HANDLE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_SR_HANDLE 0x02
```

Description

This is macro SDP_ERROR_INVALID_SR_HANDLE.

SDP_ERROR_RESERVED Macro

File

sdp.h

C

```
#define SDP_ERROR_RESERVED 0x00
```

Description

This is macro SDP_ERROR_RESERVED.

SDP_ErrorResponse Macro

File

sdp.h

C

```
#define SDP_ErrorResponse 0x01
```

Description

This is macro SDP_ErrorResponse.

SDP_FTP_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_FTP_SERVICE_ID 0x1237
```

Description

This is macro SDP_FTP_SERVICE_ID.

SDP_HID_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HID_SERVICE_ID 0x1235
```

Description

This is macro SDP_HID_SERVICE_ID.

SDP_HSP_AG_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HSP_AG_SERVICE_ID 0x1236
```

Description

This is macro SDP_HSP_AG_SERVICE_ID.

SDP_HSP_HS_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HSP_HS_SERVICE_ID 0x1236
```

Description

This is macro SDP_HSP_HS_SERVICE_ID.

SDP_MAX_ATTRIBUTE_PATTERN_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_ATTRIBUTE_PATTERN_LEN 12
```

Description

This is macro SDP_MAX_ATTRIBUTE_PATTERN_LEN.

SDP_MAX_DATA_ELEMENT_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_DATA_ELEMENT_LEN 256
```

Description

This is macro SDP_MAX_DATA_ELEMENT_LEN.

SDP_MAX_DATA_ELEMENTS Macro

File

sdp.h

C

```
#define SDP_MAX_DATA_ELEMENTS 10
```

Description

This is macro SDP_MAX_DATA_ELEMENTS.

SDP_MAX_SEARCH_PATTERN_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_SEARCH_PATTERN_LEN 12
```

Description

This is macro SDP_MAX_SEARCH_PATTERN_LEN.

SDP_MAX_TRANSACTIONS Macro

File

sdp.h

C

```
#define SDP_MAX_TRANSACTIONS 2
```

Description

typedef struct _sdp_db { bt_int count; bt_sdp_service_record_p *records; } bt_sdp_db, *bt_sdp_db_p;

SDP_PDU_HEADER_LEN Macro

File

sdp.h

C

```
#define SDP_PDU_HEADER_LEN 5
```

Description

This is macro SDP_PDU_HEADER_LEN.

SDP_RFCOMM_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_RFCOMM_SERVICE_ID 0x1234
```

Description

This is macro SDP_RFCOMM_SERVICE_ID.

SDP_ServiceAttributeRequest Macro

File

sdp.h

C

```
#define SDP_ServiceAttributeRequest 0x04
```

Description

This is macro SDP_ServiceAttributeRequest.

SDP_ServiceAttributeResponse Macro

File

sdp.h

C

```
#define SDP_ServiceAttributeResponse 0x05
```

Description

This is macro SDP_ServiceAttributeResponse.

SDP_ServiceSearchAttributeRequest Macro

File

sdp.h

C

```
#define SDP_ServiceSearchAttributeRequest 0x06
```

Description

This is macro SDP_ServiceSearchAttributeRequest.

SDP_ServiceSearchAttributeResponse Macro

File

sdp.h

C

```
#define SDP_ServiceSearchAttributeResponse 0x07
```

Description

This is macro SDP_ServiceSearchAttributeResponse.

SDP_ServiceSearchRequest Macro

File

sdp.h

C

```
#define SDP_ServiceSearchRequest 0x02
```

Description

This is macro SDP_ServiceSearchRequest.

SDP_ServiceSearchResponse Macro

File

sdp.h

C

```
#define SDP_ServiceSearchResponse 0x03
```

Description

This is macro SDP_ServiceSearchResponse.

SDP_SR_HANDLE_HDP_SINK Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HDP_SINK 0x10009
```

Description

This is macro SDP_SR_HANDLE_HDP_SINK.

SDP_SR_HANDLE_HDP_SOURCE Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HDP_SOURCE 0x10008
```

Description

This is macro SDP_SR_HANDLE_HDP_SOURCE.

SDP_SR_HANDLE_HFP_HF Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HFP_HF 0x1000A
```

Description

This is macro SDP_SR_HANDLE_HFP_HF.

SDP_SR_HANDLE_HID Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HID 0x10001
```

Description

This is macro SDP_SR_HANDLE_HID.

SDP_SR_HANDLE_HID_KEYBOARD Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HID_KEYBOARD 0x10005
```

Description

This is macro SDP_SR_HANDLE_HID_KEYBOARD.

SDP_SR_HANDLE_HSP_HS Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HSP_HS 0x10004
```

Description

This is macro SDP_SR_HANDLE_HSP_HS.

SDP_SR_HANDLE_OBEXFileTransfer Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_OBEXFileTransfer 0x10006
```

Description

This is macro SDP_SR_HANDLE_OBEXFileTransfer.

SDP_SR_HANDLE_OBEXObjectPush Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_OBEXObjectPush 0x10007
```

Description

This is macro SDP_SR_HANDLE_OBEXObjectPush.

SDP_SR_HANDLE_PNPINFORMATION Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_PNPINFORMATION 0x10002
```

Description

This is macro SDP_SR_HANDLE_PNPINFORMATION.

SDP_SR_HANDLE_RFCOMM Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_RFCOMM 0x10000
```

Description

This is macro SDP_SR_HANDLE_RFCOMM.

SDP_SR_HANDLE_SERVER Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_SERVER 0x0
```

Description

This is macro SDP_SR_HANDLE_SERVER.

SDP_SR_HANDLE_TEST Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_TEST 0x10003 // for simulating service search using continuation state
```

Description

for simulating service search using continuation state

bt_sdp_client_evt_connected_t Structure

File

sdp_client.h

C

```
struct bt_sdp_client_evt_connected_t {
  bt_l2cap_channel_t* channel;
};
```

Description

This is record bt_sdp_client_evt_connected_t.

bt_sdp_client_evt_disconnected_t Structure

File

sdp_client.h

C

```
struct bt_sdp_client_evt_disconnected_t {
  bt_l2cap_channel_t* channel;
};
```

Description

This is record bt_sdp_client_evt_disconnected_t.

bt_sdp_client_callback_fp Type

File

sdp_client.h

C

```
typedef void (* bt_sdp_client_callback_fp)(bt_byte evt, void* evt_param, void* param);
```

Description

This is type bt_sdp_client_callback_fp.

bt_sdp_data_element_cp Type

File

sdp.h

C

```
typedef const bt_sdp_data_element_t* bt_sdp_data_element_cp;
```

Description

This is type bt_sdp_data_element_cp.

bt_sdp_data_element_p Type

File

sdp.h

C

```
typedef bt_sdp_data_element_t* bt_sdp_data_element_p;
```

Description

This is type bt_sdp_data_element_p.

bt_sdp_data_element_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_data_element_t {
  bt_int type;
  bt_int bytecount;
  union {
    bt_ulong init;
    void* pdata;
    char* pstr;
    char* purl;
    struct _bt_sdp_sequence_t* pseq;
    bt_byte b;
    bt_int i;
    bt_uint ui;
    bt_long l;
    bt_ulong ul;
    bt_uuid16 uuid16;
    bt_uuid32 uuid32;
    bt_uuid_p uuid128;
  } data;
} bt_sdp_data_element_t;
```

Description

This is type bt_sdp_data_element_t.

bt_sdp_found_attr_list_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_found_attr_list_t {
  bt_int found_attr_count;
  const bt_byte** found_attr_list;
} bt_sdp_found_attr_list_t;
```

Members

Members	Description
const bt_byte** found_attr_list;	SDP_MAX_ATTRIBUTE_RESULT_LEN];

Description

This is type bt_sdp_found_attr_list_t.

bt_sdp_packet_t Structure

File

sdp_packet.h

C

```
typedef struct _bt_sdp_packet_t {
  bt_packet_t header;
  bt_byte sdu_type;
  union {
    union {
      struct _bt_sdp_service_transaction_t* service;
      struct _bt_sdp_transaction_t* attribute;
    } transaction;
    struct {
      bt_int trans_id;
      bt_int err_code;
    } error;
    struct {
      bt_int trans_id;
      struct _bt_sdp_data_element_t* pattern;
      bt_int uuid_index;
      bt_int uuid_pos;
    } service_search;
    struct {
      bt_int trans_id;
      struct _bt_sdp_data_element_t* pattern;
      bt_long sr;
    } attribute_search;
  } data;
} bt_sdp_packet_t;
```

Description

This is type bt_sdp_packet_t.

bt_sdp_read_de_callback_ftp Type

File

sdp_client.h

C

```
typedef bt_bool (* bt_sdp_read_de_callback_ftp)(bt_sdp_data_element_p pde, bt_bool seq_start);
```

Description

This is type bt_sdp_read_de_callback_ftp.

bt_sdp_sequence_cp Type

File

sdp.h

C

```
typedef const bt_sdp_sequence_t* bt_sdp_sequence_cp;
```

Description

This is type bt_sdp_sequence_cp.

bt_sdp_sequence_p Type

File

sdp.h

C

```
typedef bt_sdp_sequence_t* bt_sdp_sequence_p;
```

Description

This is type bt_sdp_sequence_p.

bt_sdp_sequence_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_sequence_t {
  bt_int count;
  bt_sdp_data_element_p elements;
} bt_sdp_sequence_t;
```

Description

This is type bt_sdp_sequence_t.

bt_sdp_serialization_state_p Structure

File

sdp.h

C

```
typedef struct _bt_sdp_serialization_state_t {
  bt_int listIndex;
  bt_int attrIndex;
  bt_byte attrPart;
  bt_int attrPartPos;
  bt_int attr_bytes;
  bt_int first_list;
  bt_int last_list;
  bt_int first_attr;
  bt_int last_attr;
  bt_uint attr_lists_seq_len;
  bt_bool write_seq_len;
} bt_sdp_serialization_state_t, * bt_sdp_serialization_state_p;
```

Description

This is type bt_sdp_serialization_state_p.

bt_sdp_serialization_state_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_serialization_state_t {
  bt_int listIndex;
  bt_int attrIndex;
  bt_byte attrPart;
  bt_int attrPartPos;
  bt_int attr_bytes;
  bt_int first_list;
  bt_int last_list;
  bt_int first_attr;
  bt_int last_attr;
  bt_uint attr_lists_seq_len;
  bt_bool write_seq_len;
} bt_sdp_serialization_state_t, * bt_sdp_serialization_state_p;
```

Description

This is type bt_sdp_serialization_state_t.

bt_sdp_server_attribute_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_attribute_t {
  bt_uint attr_id;
  bt_uint data_len;
  const bt_byte* data;
  bt_sdp_server_data_element_t attr_value;
} bt_sdp_server_attribute_t;
```

Description

This is type bt_sdp_server_attribute_t.

bt_sdp_server_data_element_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_data_element_t {
  bt_byte type;
  bt_uint data_len;
  const bt_byte* data;
} bt_sdp_server_data_element_t;
```

Description

Private types

bt_sdp_server_record_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_record_t {
  bt_sr_handle_t h;
```

```
  bt_uint data_len;
  const bt_byte* data;
} bt_sdp_server_record_t;
```

Description

This is type bt_sdp_server_record_t.

bt_sdp_service_attribute_callback_fp Type

File

sdp_client.h

C

```
typedef bt_bool (* bt_sdp_service_attribute_callback_fp)(bt_uint attrid, bt_sdp_data_element_p pde, void*
param);
```

Description

This is type bt_sdp_service_attribute_callback_fp.

bt_sdp_service_search_callback_fp Type

File

sdp_client.h

C

```
typedef void (* bt_sdp_service_search_callback_fp)(bt_sr_handle_p shlist, bt_int count, void* param);
```

Description

This is type bt_sdp_service_search_callback_fp.

bt_sdp_service_transaction_p Structure

File

sdp.h

C

```
typedef struct _bt_sdp_service_transaction_t {
  bt_int id;
  bt_int found_sr_count;
  bt_sr_handle_t* found_sr_list;
  bt_uint max_sr_count;
  bt_int next_tran_index;
} bt_sdp_service_transaction_t, * bt_sdp_service_transaction_p;
```

Members

Members	Description
bt_sr_handle_t* found_sr_list;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_service_transaction_p.

bt_sdp_service_transaction_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_service_transaction_t {
  bt_int id;
  bt_int found_sr_count;
  bt_sr_handle_t* found_sr_list;
```

```
  bt_uint max_sr_count;
  bt_int next_tran_index;
} bt_sdp_service_transaction_t, * bt_sdp_service_transaction_p;
```

Members

Members	Description
bt_sr_handle_t* found_sr_list;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_service_transaction_t.

bt_sdp_transaction_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_transaction_t {
  bt_int id;
  bt_int found_list_count;
  bt_sdp_found_attr_list_t* found_attr_lists;
  bt_uint max_bytecount;
  bt_bool complete;
  bt_bool continuation;
  bt_sdp_serialization_state_t sr_state;
} bt_sdp_transaction_t;
```

Members

Members	Description
bt_sdp_found_attr_list_t* found_attr_lists;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_transaction_t.

bt_sr_handle_p Type

File

sdp.h

C

```
typedef bt_long * bt_sr_handle_p;
```

Description

This is type bt_sr_handle_p.

bt_sr_handle_t Type

File

sdp.h

C

```
typedef bt_long bt_sr_handle_t;
```

Description

This is type bt_sr_handle_t.

SDP_CLIENT_EVT_CONNECTION_FAILED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_CONNECTION_FAILED 3
```

Description

This is macro SDP_CLIENT_EVT_CONNECTION_FAILED.

SDP_ATTRID_HCRP_DeviceLocation Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_DeviceLocation 0x306
```

Description

This is macro SDP_ATTRID_HCRP_DeviceLocation.

SDP_ATTRID_HCRP_DeviceName Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_DeviceName 0x302
```

Description

This is macro SDP_ATTRID_HCRP_DeviceName.

SDP_ATTRID_HCRP_FriendlyName Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_FriendlyName 0x304
```

Description

This is macro SDP_ATTRID_HCRP_FriendlyName.

SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT 0x1125
```

Description

This is macro SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT.

SDP_CLSID_HARD_COPY_CONTROL_CHANNEL Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_CONTROL_CHANNEL 0x0012
```

Description

This is macro SDP_CLSID_HARD_COPY_CONTROL_CHANNEL.

SDP_CLSID_HARD_COPY_DATA_CHANNEL Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_DATA_CHANNEL 0x0014
```

Description

This is macro SDP_CLSID_HARD_COPY_DATA_CHANNEL.

SDP_CLSID_HARD_COPY_NOTIFICATION Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_NOTIFICATION 0x0016
```

Description

This is macro SDP_CLSID_HARD_COPY_NOTIFICATION.

SDP_CLSID_HCR_PRINT Macro

File

sdp.h

C

```
#define SDP_CLSID_HCR_PRINT 0x1126
```

Description

This is macro SDP_CLSID_HCR_PRINT.

SDP_CLSID_HCR_SCAN Macro

File

sdp.h

C

```
#define SDP_CLSID_HCR_SCAN 0x1127
```

Description

This is macro SDP_CLSID_HCR_SCAN.

SDP_ATTRID_HCRP_1284ID Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_1284ID 0x300
```

Description

This is macro SDP_ATTRID_HCRP_1284ID.

SPP Functions

bt_spp_allocate Function

File

spp.h

C

```
bt_spp_port_t* bt_spp_allocate(bt_l2cap_mgr_t* l2cap_mgr, bt_spp_state_callback_fp callback, void*
callback_param);
```

Description

- Allocate a serial port.

The returned serial port is initially in the ::SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp_listen(). The p callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the ::SPP_MAX_PORTS configuration parameter.

@param l2cap_mgr L2CAP manager.

@param callback Pointer to a callback function used to notify about serial port events. Cannot be NULL.

@param callback_param An arbitrary pointer that is passed as a parameter to the callback function.

@return A pointer to the ::bt_spp_port_t structure. Returns NULL if the maximum number of ports has been already allocated or the c callback parameter is NULL.

bt_spp_cancel_receive Function

File

spp.h

C

```
void bt_spp_cancel_receive(bt_spp_port_t* port);
```

Description

- Cancel receive data.

If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called.

If there is no receive operation in progress calling this function has no effect.

@param port Serial port.

bt_spp_cancel_send Function

File

spp.h

C

```
void bt_spp_cancel_send(bt_spp_port_t* port);
```

Description

- Cancel send data.

If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the ::SPP_SEND_STATUS_CANCELED status.

If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the ::SPP_SEND_STATUS_CANCELED status.

If there is no send operation in progess calling this function has no effect.

@param port Serial port.

bt_spp_clr_port_options Function

File

spp.h

C

```
void bt_spp_clr_port_options(bt_spp_port_t* port, bt_uint options);
```

Description

This is function bt_spp_clr_port_options.

bt_spp_connect Function

File

spp.h

C

```
bt_bool bt_spp_connect(bt_spp_port_t* port, bt_bdaddr_p remote_addr, bt_byte channel);
```

Description

- Connect to a remote device.

This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the ::SPP_PORT_EVENT_CONNECTION_FAILED event.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

@param port Serial port. @param remote_addr Bluetooth address of the remote device. @param channel RFCOMM server channel on which the connection is to be established.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_deallocate Function

File

spp.h

C

```
bt_bool bt_spp_deallocate(bt_spp_port_t* port);
```

Description

- Deallocate serial port.

This function deallocates the specified port structure and other resources associated with it.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate().

@param port Serial port structure to deallocate.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_disconnect Function

File

spp.h

C

```
void bt_spp_disconnect(bt_spp_port_t* port);
```

Description

- Disconnect from the remote device.

This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event.

If the port is already in the disconnected state the function does nothing and the callback is not called.

@param port Serial port.

bt_spp_find_server Function

File

spp.h

C

```
bt_bool bt_spp_find_server(bt_bdaddr_t* deviceAddress, const bt_uuid_t* service_class_id,
bt_spp_find_server_callback_fp callback, void* callback_param);
```

Description

This is function bt_spp_find_server.

bt_spp_get_frame_length Function

File

spp.h

C

```
bt_int bt_spp_get_frame_length(bt_spp_port_t* port);
```

Description

- Get frame length.

This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length.

@return RFCOMM frame length in bytes.

bt_spp_get_hci_connection Function

File

spp.h

C

```
bt_hci_conn_state_t* bt_spp_get_hci_connection(const bt_spp_port_t* port);
```

Description

- Get SPP port's ACL connection.

This function returns a pinter to the structure that describes the ACL connection this port is on.

@return Pointer to ACL connection description if the port is connected, NULL otherwise.

bt_spp_get_local_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_get_local_modem_status(const bt_spp_port_t* port);
```

Description

- Get local device's TS 07.10 control signals.

This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD

@param port Serial port.

@return local device's TS 07.10 control signals.

bt_spp_get_remote_address Function

File

spp.h

C

```
bt_bdaddr_t* bt_spp_get_remote_address(const bt_spp_port_t* port);
```

Description

brief Get the address of the remote device this device is connected to. ingroup spp

param port Serial port.

return li c A pointer to bt_bdaddr structure that contains the address of the remote device.

bt_spp_get_remote_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_get_remote_modem_status(const bt_spp_port_t* port);
```

Description

- Get remote device's TS 07.10 control signals.

This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD

@param port Serial port.

@return remote device's TS 07.10 control signals.

bt_spp_init Function

File

spp.h

C

```
void bt_spp_init();
```

Description

- Initialize the SPP module.

This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.

bt_spp_listen Function

File

spp.h

C

```
bt_bool bt_spp_listen(bt_spp_port_t* port, bt_byte channel);
```

Description

- Listen for incoming connections.

This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use.

When a remote device successfully establishes a connection on the specified port the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

@param port Serial port. @param channel The RFCOMM server channel on which to listen for connections.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_receive Function

File

spp.h

C

```
bt_bool bt_spp_receive(bt_spp_port_t* port, void* buffer, bt_int buffer_len, bt_spp_receive_callback_fp
callback);
```

Description

- Receive data.

This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation.

The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started receive operation is still in progress.

@param port Serial port. @param data Pointer to the data to be sent. @param data_len Length of the data. @param callback Send callback function.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_send Function

File

spp.h

C

```
bt_bool bt_spp_send(bt_spp_port_t* port, const void* data, bt_ulong data_len, bt_spp_send_callback_fp
callback);
```

Description

- Send data.

This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the ::SPP_PORT_EVENT_SEND_PROGRESS event.

The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress.

The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate().

@param port Serial port. @param data Pointer to the data to be sent. @param data_len Length of the data. @param callback Send callback function.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_dtr Function

File

spp.h

C

```
bt_byte bt_spp_set_dtr(bt_spp_port_t* port, bt_bool on);
```

Description

- Set local device's RS-232 DTR signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param on Value indicating weather to set or clear the signal.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_local_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_set_local_modem_status(bt_spp_port_t* port, bt_byte ms);
```

Description

- Set local device's TS 07.10 control signals.

Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param ms Signals mask.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_port_options Function

File

spp.h

C

```
void bt_spp_set_port_options(bt_spp_port_t* port, bt_uint options);
```

Description

This is function bt_spp_set_port_options.

bt_spp_set_rts Function

File

spp.h

C

```
bt_byte bt_spp_set_rts(bt_spp_port_t* port, bt_bool on);
```

Description

- Set local device's RS-232 RTS signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param on Value indicating weather to set or clear the signal.

@return c TRUE if successful, c FALSE otherwise.

_bt_spp_find_port Function

File

spp_private.h

C

```
bt_spp_port_t* _bt_spp_find_port(bt_rfcomm_dlc_t* dlc);
```

Description

This is function _bt_spp_find_port.

_bt_spp_handle_rx Function

File

spp_private.h

C

```
void _bt_spp_handle_rx(bt_spp_port_t* port);
```

Description

This is function _bt_spp_handle_rx.

_bt_spp_handle_tx Function

File

spp_private.h

C

```
void _bt_spp_handle_tx(bt_spp_port_t* port);
```

Description

This is function _bt_spp_handle_tx.

_bt_spp_rfcomm_read_data_callback Function

File

spp_private.h

C

```
void _bt_spp_rfcomm_read_data_callback(bt_rfcomm_dlc_p pdlc, bt_byte_p pdata, bt_int len);
```

Description

This is function _bt_spp_rfcomm_read_data_callback.

_bt_spp_rfcomm_send_data_callback Function

File

spp_private.h

C

```
void _bt_spp_rfcomm_send_data_callback(bt_rfcomm_dlc_t* dlc, bt_byte* data, bt_int len, bt_int status);
```

Description

This is function _bt_spp_rfcomm_send_data_callback.

_bt_spp_client_init Function

File

spp_private.h

C

```
void _bt_spp_client_init();
```

Description

This is function _bt_spp_client_init.

bt_spp_cancel_listen Function

File

spp.h

C

```
bt_bool bt_spp_cancel_listen(bt_spp_port_t* port, bt_byte server_channel);
```

Description

- Stop listening for incoming connections.

This function stops the port to accept incoming connections on a given server channel.

@param port Serial port. @param channel The RFCOMM server channel to accept connections on.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_find_server_ex Function

File

spp.h

C

```
bt_bool bt_spp_find_server_ex(bt_bdaddr_t* deviceAddress, const bt_uuid_t* service_class_id, bt_bool
force_hci_disconnect, bt_spp_find_server_callback_fp callback, bt_sdp_client_callback_fp client_callback,
void* callback_param);
```

Description

This is function bt_spp_find_server_ex.

SPP Data Types and Constants

SPP_PORT_OPTION_ENCRYPTED Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_ENCRYPTED 0x0002
```

Description

This is macro SPP_PORT_OPTION_ENCRYPTED.

SPP_PORT_OPTION_MASTER Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_MASTER 0x0004
```

Description

This is macro SPP_PORT_OPTION_MASTER.

SPP_PORT_OPTION_SECURE Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_SECURE 0x0001
```

Description

This is macro SPP_PORT_OPTION_SECURE.

SPP_PORT_TYPE_INCOMING Macro

File

spp.h

C

```
#define SPP_PORT_TYPE_INCOMING 0
```

Description

This is macro SPP_PORT_TYPE_INCOMING.

SPP_PORT_TYPE_OUTGOING Macro

File

spp.h

C

```
#define SPP_PORT_TYPE_OUTGOING 1
```

Description

This is macro SPP_PORT_TYPE_OUTGOING.

SPP_RS232_CTS Macro

File

spp.h

C

```
#define SPP_RS232_CTS RFCOMM_MODEM_STATUS_RTR
```

Description

This is macro SPP_RS232_CTS.

SPP_RS232_DCD Macro

File

spp.h

C

```
#define SPP_RS232_DCD RFCOMM_MODEM_STATUS_DV
```

Description

This is macro SPP_RS232_DCD.

SPP_RS232_DSR Macro

File

spp.h

C

```
#define SPP_RS232_DSR RFCOMM_MODEM_STATUS_RTC
```

Description

- @defgroup spp Serial Port Profile (SPP)

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

Here are the steps for using this API:

- Call the bt_spp_init() function.
- Allocate a serial port structure with bt_spp_allocate(). One of the parameters to

this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes.

- To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state callback function.
- To wait for a connection from a remote device call bt_spp_listen(). The stack will notify when the connection is established by calling the state callback function.
- When the port is connected you can send data with bt_spp_send() and receive data with bt_spp_receive().
- To terminate the connection call bt_spp_disconnect().
- When you are finished using the port deallocate it with bt_spp_deallocate().

SPP_RS232_DTR Macro

File

spp.h

C

```
#define SPP_RS232_DTR RFCOMM_MODEM_STATUS_RTC
```

Description

This is macro SPP_RS232_DTR.

SPP_RS232_RI Macro

File

spp.h

C

```
#define SPP_RS232_RI RFCOMM_MODEM_STATUS_IC
```

Description

This is macro SPP_RS232_RI.

SPP_RS232_RTS Macro

File

spp.h

C

```
#define SPP_RS232_RTS RFCOMM_MODEM_STATUS_RTR
```

Description

This is macro SPP_RS232_RTS.

bt_spp_find_server_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_find_server_callback_fp)(bt_byte server_channel, bt_bool found, void* param);
```

Description

This is type bt_spp_find_server_callback_fp.

bt_spp_port_event_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_port_event_e {
  SPP_PORT_EVENT_CONNECTED = 1,
  SPP_PORT_EVENT_DISCONNECTED,
  SPP_PORT_EVENT_CONNECTION_FAILED,
  SPP_PORT_EVENT_SEND_PROGRESS,
  SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED,
  SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED,
  SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED
} bt_spp_port_event_e;
```

Members

Members	Description
SPP_PORT_EVENT_CONNECTED = 1	• Connection with a remote device was successfully established. This event is reported independent of which side (local or remote) initiated the connection.
SPP_PORT_EVENT_DISCONNECTED	• Connection with the remote device was terminated. This event is reported independent of which side (local or remote) initiated termination of the connection.
SPP_PORT_EVENT_CONNECTION_FAILED	• Connection to a remote device failed. This event is reported when an attempt to establish a connection using bt_spp_connect() has failed.
SPP_PORT_EVENT_SEND_PROGRESS	• Send operation progress. This event is reported periodically during sending data over the serial port connection. It can be used by the application to track send operation progress.
SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED	• Remote modem status. This event is reported when a remote device notifies the local device about the status of its V.24 control signals. The actual value of the signals can be obtained by calling bt_spp_get_remote_modem_status(const bt_spp_port_t* port)
SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED	• Local modem status. This event is reported when the local device has successfully sent the state of its V.24 signals to the remote party.
SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED	• Local modem status. This event is reported when the local device failed to send the state of its V.24 signals to the remote party.

Description

- Serial port event.

Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().

bt_spp_port_state_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_port_state_e {
  SPP_PORT_STATE_FREE,
  SPP_PORT_STATE_DISCONNECTED,
  SPP_PORT_STATE_DISCONNECTING,
  SPP_PORT_STATE_CONNECTING,
  SPP_PORT_STATE_CONNECTED
} bt_spp_port_state_e;
```

Members

Members	Description
SPP_PORT_STATE_FREE	*< Used internally to mark a port structure as available for allocation. The application should never encounter a port in this state.
SPP_PORT_STATE_DISCONNECTED	< Port is not connected.
SPP_PORT_STATE_DISCONNECTING	< Disconnecting from remote device.
SPP_PORT_STATE_CONNECTING	< Connecting to a remote device.
SPP_PORT_STATE_CONNECTED	< Port is connected to a remote device.

Description

- Serial port state.

Values of this enumeration represent states of the serial port.

bt_spp_port_t Type

File

spp.h

C

```
typedef struct _bt_spp_port_t bt_spp_port_t;
```

Description

This is type bt_spp_port_t.

bt_spp_receive_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_receive_callback_fp)(bt_spp_port_t* port, bt_int bytes_received, void* param);
```

Description

- Serial port receive callback.

This callback function is called when a receive operation initiated by bt_spp_receive() completes.

@param port Serial port on which the receive operation completed.

@param bytes_received Number of received bytes.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

bt_spp_send_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_send_callback_fp)(bt_spp_port_t* port, bt_ulong bytes_sent, bt_spp_send_status_e
status, void* param);
```

Description

- Serial port send callback.

This callback function is called when a send operation initiated by bt_spp_send() completes.

@param port Serial port on which the send operation completed.

@param bytes_sent Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send();

@param status Completion status. It is one of the values defined in the ::bt_spp_send_status_e enumeration.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

bt_spp_send_status_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_send_status_e {
  SPP_SEND_STATUS_SUCCESS = 0,
  SPP_SEND_STATUS_TIMEOUT,
  SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES,
  SPP_SEND_STATUS_CANCELED,
  SPP_SEND_STATUS_INTERRUPTED
} bt_spp_send_status_e;
```

Members

Members	Description
SPP_SEND_STATUS_SUCCESS = 0	< The operation completed successfully.
SPP_SEND_STATUS_TIMEOUT	< The operation timed out.
SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES	< There was not enough resources to complete the operation.
SPP_SEND_STATUS_CANCELED	< The operation was canceled.
SPP_SEND_STATUS_INTERRUPTED	< The operation was interrupted because the connection closed.

Description

Send operation status.

bt_spp_state_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_state_callback_fp)(bt_spp_port_t* port, bt_spp_port_event_e evt, void* param);
```

Description

- Serial port state callback.

This callback function is called whenever the state of a serial port is changed.

@param port Serial port which state has changed.

@param evt Event describing the nature of state change. It is one of the values defined in the ::bt_spp_port_event_e enumeration.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

SSP Functions

bt_ssp_evt_handler Function

File

ssp.h

C

```
void bt_ssp_evt_handler(bt_hci_event_t* evt);
```

Description

This is function bt_ssp_evt_handler.

bt_ssp_init Function

File

ssp.h

C

```
void bt_ssp_init();
```

Description

This is function bt_ssp_init.

bt_ssp_read_local_oob_data Function

File

ssp.h

C

```
bt_bool bt_ssp_read_local_oob_data(bt_spp_read_local_oob_data_callback_fp callback);
```

Description

This is function bt_ssp_read_local_oob_data.

bt_ssp_send_keypress_notification Function

File

ssp.h

C

```
bt_bool bt_ssp_send_keypress_notification(bt_bdaddr_p bdaddr_remote, SSP_KEYPRESS_NOTIFICATION_TYPE
keypress_type);
```

Description

This is function bt_ssp_send_keypress_notification.

bt_ssp_send_user_confirmation Function

File

ssp.h

C

```
bt_bool bt_ssp_send_user_confirmation(bt_byte status, bt_ssp_user_confirmation_request* ucr,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_ssp_send_user_confirmation.

bt_ssp_send_user_passkey Function

File

ssp.h

C

```
bt_bool bt_ssp_send_user_passkey(bt_byte status, bt_ssp_user_passkey_request* upkr, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_ssp_send_user_passkey.

bt_ssp_set_io_capabilities Function

File

ssp.h

C

```
bt_bool bt_ssp_set_io_capabilities(bt_byte status, bt_ssp_io_capability* io_caps, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_ssp_set_io_capabilities.

bt_ssp_set_mode Function

File

ssp.h

C

```
bt_bool bt_ssp_set_mode(SSP_MODE mode, bt_hci_cmd_callback_fp callback);
```

Description

This is function bt_ssp_set_mode.

bt_ssp_set_oob_data Function

File

ssp.h

C

```
bt_bool bt_ssp_set_oob_data(bt_byte status, bt_ssp_oob_data* oob_data, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_ssp_set_oob_data.

ssp_evt_io_capability_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_io_capability_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_io_capability_request.

ssp_evt_io_capability_response Function

File

ssp_event_handler.h

C

```
void ssp_evt_io_capability_response(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_io_capability_response.

ssp_evt_keypress_notification Function

File

ssp_event_handler.h

C

```
void ssp_evt_keypress_notification(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_keypress_notification.

ssp_evt_oob_data_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_oob_data_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_oob_data_request.

ssp_evt_ssp_complete Function

File

ssp_event_handler.h

C

```
void ssp_evt_ssp_complete(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_ssp_complete.

ssp_evt_user_confirmation_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_confirmation_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_confirmation_request.

ssp_evt_user_passkey_notification Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_passkey_notification(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_passkey_notification.

ssp_evt_user_passkey_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_passkey_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_passkey_request.

SSP Data Types and Constants

OOB_DATA_HASH_LENGTH Macro

File

ssp.h

C

```
#define OOB_DATA_HASH_LENGTH 16
```

Description

Remote OOB Data Request Event

OOB_DATA_RANDOMIZER_LENGTH Macro

File

ssp.h

C

```
#define OOB_DATA_RANDOMIZER_LENGTH 16
```

Description

This is macro OOB_DATA_RANDOMIZER_LENGTH.

SSP_MAX_MANAGERS Macro

File

ssp.h

C

```
#define SSP_MAX_MANAGERS 1
```

Description

This is macro SSP_MAX_MANAGERS.

_bt_spp_port_t Structure

File

spp.h

C

```
struct _bt_spp_port_t {
  bt_byte state;
  bt_byte type;
  bt_byte server_channel;
  bt_rfcomm_dlc_t* rfcomm_dlc;
  bt_byte cur_local_ms;
  bt_ulong listening_server_channels;
  struct _bt_spp_port_flags_t {
    unsigned int sig_tx : 1;
    unsigned int sig_rx : 1;
    unsigned int sig_state : 1;
    unsigned int sig_credit : 1;
    unsigned int open : 1;
    unsigned int listening : 1;
    unsigned int secure : 1;
    unsigned int encrypted : 1;
    unsigned int master : 1;
  } flags;
  bt_signal_t signal;
  bt_spp_state_callback_fp state_cb;
  bt_spp_receive_callback_fp receive_cb;
  bt_spp_send_callback_fp send_cb;
  void* cb_param;
  bt_byte* rx_buffer;
  bt_int rx_buffer_len;
  bt_byte* tx_data;
  bt_ulong tx_data_len;
  bt_ulong tx_remaining_len;
  bt_bool tx_cancel;
  bt_byte remaining_connect_attemtps;
  bt_bdaddr_t connect_address;
  bt_rfcomm_session_listener_t rfcomm_session_listener;
  bt_byte* frame_buffers;
  bt_int* frame_len;
  bt_byte read_frame_index;
  bt_byte write_frame_index;
  bt_int read_frame_pos;
};
```

Members

Members	Description
bt_byte state;	• Port state. The field is set to one of the values defined in the ::bt_spp_port_state_e enumeration. This field must never be modified by the application.

Description

- Serial port structure.

This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

bt_spp_read_local_oob_data_callback_fp Type

File

ssp.h

C

```
typedef void (* bt_spp_read_local_oob_data_callback_fp)(bt_byte status, bt_ssp_oob_data* oob_data, void*
init_param);
```

Description

This is type bt_spp_read_local_oob_data_callback_fp.

bt_ssp_io_capability Structure

File

ssp.h

C

```
typedef struct _bt_ssp_io_capability {
  bt_bdaddr_t bdaddr_remote;
  SSP_IO_CAPABILITY io_capability;
  SSP_OOB_DATA_PRESENT oob_data_present;
  SSP_AUTHENTICATION_REQUIREMENTS authentication_requirements;
} bt_ssp_io_capability;
```

Description

This is type bt_ssp_io_capability.

bt_ssp_keypress_notification Structure

File

ssp.h

C

```
typedef struct _bt_ssp_keypress_notification {
  bt_bdaddr_t bdaddr_remote;
  SSP_KEYPRESS_NOTIFICATION_TYPE type;
} bt_ssp_keypress_notification;
```

Description

Keypress Notification Event

bt_ssp_oob_data Structure

File

ssp.h

C

```
typedef struct _bt_ssp_oob_data {
  bt_bdaddr_t bdaddr;
  bt_byte hash[OOB_DATA_HASH_LENGTH];
  bt_byte randomizer[OOB_DATA_RANDOMIZER_LENGTH];
} bt_ssp_oob_data;
```

Description

This is type bt_ssp_oob_data.

bt_ssp_simple_pairing_complete Structure

File

ssp.h

C

```
typedef struct _bt_ssp_simple_pairing_complete {
  bt_byte status;
  bt_bdaddr_t bdaddr_remote;
} bt_ssp_simple_pairing_complete;
```

Description

Simple Pairing Complete Event

bt_ssp_user_confirmation_request Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_confirmation_request {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong numeric_value;
} bt_ssp_user_confirmation_request;
```

Description

User Confirmation Request Event

bt_ssp_user_passkey_notification Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_passkey_notification {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong passkey;
} bt_ssp_user_passkey_notification;
```

Description

User Passkey Notification Event

bt_ssp_user_passkey_request Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_passkey_request {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong passkey;
} bt_ssp_user_passkey_request;
```

Description

User Passkey Request Event

SSP_AUTHENTICATION_REQUIREMENTS Enumeration

File

ssp.h

C

```
typedef enum _SSP_AUTHENTICATION_REQUIREMENTS {
  SSP_MITM_NOT_REQUIRED_NO_BONDING = 0,
  SSP_MITM_REQUIRED_NO_BONDING,
  SSP_MITM_NOT_REQUIRED_DEDICATED_BONDING,
  SSP_MITM_REQUIRED_DEDICATED_BONDING,
  SSP_MITM_NOT_REQUIRED_GENERAL_BONDING,
  SSP_MITM_REQUIRED_GENERAL_BONDING
} SSP_AUTHENTICATION_REQUIREMENTS;
```

Description

This is type SSP_AUTHENTICATION_REQUIREMENTS.

SSP_IO_CAPABILITY Enumeration

File

ssp.h

C

```
typedef enum _SSP_IO_CAPABILITY {
  SSP_IO_CAPABILITY_DISPLAY_ONLY = 0,
  SSP_IO_CAPABILITY_DISPLAY_YESNO,
  SSP_IO_CAPABILITY_KEYBOARD_ONLY,
  SSP_IO_CAPABILITY_NO_INPUT_NO_OUTPUT
} SSP_IO_CAPABILITY;
```

Description

This is type SSP_IO_CAPABILITY.

SSP_KEYPRESS_NOTIFICATION_TYPE Enumeration

File

ssp.h

C

```
typedef enum _SSP_KEYPRESS_NOTIFICATION_TYPE {
  PASSKEY_ENTRY_STYARTED,
  PASSKEY_DIGIT_ENTERED,
  PASSKEY_DIGIT_ERASED,
  PASSKEY_CLEARED,
  PASSKEY_ENTRY_COMPLETED
} SSP_KEYPRESS_NOTIFICATION_TYPE;
```

Description

This is type SSP_KEYPRESS_NOTIFICATION_TYPE.

SSP_MODE Enumeration

File

ssp.h

C

```
typedef enum _SSP_MODE {
  SSP_MODE_DISABLED = 0,
  SSP_MODE_ENABLED
} SSP_MODE;
```

Description

This is type SSP_MODE.

SSP_OOB_DATA_PRESENT Enumeration

File

ssp.h

C

```
typedef enum _SSP_OOB_DATA_PRESENT {
  SSP_OOB_DATA_NOT_PRESENT = 0,
  SSP_OOB_DATA_REMOTE_DEVICE_DATA_PRESENT
} SSP_OOB_DATA_PRESENT;
```

Description

This is type SSP_OOB_DATA_PRESENT.

SSP_EVENT Enumeration

File

ssp_event.h

C

```
typedef enum _SSP_EVENT {
  SSP_EVENT_IO_CAPABILITY_RESPONSE,
  SSP_EVENT_KEYPRESS_NOTIFICATION,
  SSP_EVENT_USER_PASSKEY_REQUEST,
  SSP_EVENT_USER_PASSKEY_NOTIFICATION,
  SSP_EVENT_USER_CONFIRMATION_REQUEST,
  SSP_EVENT_OOB_DATA_REQUEST,
  SSP_EVENT_IO_CAPABILITY_REQUEST,
  SSP_EVENT_SIMPLE_PAIRING_COMPLETE
} SSP_EVENT;
```

Description

This is type SSP_EVENT.

Misc. Functions

btx_csr_alloc_bccmd_getreq Function

File

csr.h

C

```
bt_hci_command_t* btx_csr_alloc_bccmd_getreq(bt_uint var_id, bt_uint data_word_count,
bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_alloc_bccmd_getreq.

btx_csr_alloc_bccmd_setreq Function

File

csr.h

C

```
bt_hci_command_t* btx_csr_alloc_bccmd_setreq(bt_uint var_id, bt_uint data_word_count,
bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_alloc_bccmd_setreq.

btx_csr_autobaud Function

File

csr.h

C

```
void btx_csr_autobaud(btx_csr_autobaud_buffer_t* buffer, btx_csr_autobaud_callback_fp callback, void*
callback_param);
```

Description

brief Configure controller's UART speed. ingroup btx_csr

details This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.

btx_csr_bc7_sel_host_interface_h4 Function

File

csr.h

C

```
void btx_csr_bc7_sel_host_interface_h4(btx_csr_autobaud_buffer_t* buffer, bt_byte interval,
btx_csr_autobaud_callback_fp callback, void* callback_param);
```

Description

brief Configure controller's UART speed and host interface. ingroup btx_csr

details This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HOST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.

btx_csr_enable_tx Function

File

csr.h

C

```
bt_bool btx_csr_enable_tx(bt_bool enable, bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Enable/disable transmitter ingroup btx_csr

param enable Specifies whether the transmitter should be enable or disabled. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_exec_script Function

File

csr.h

C

```
void btx_csr_exec_script(const btx_csr_script_t* script, btx_csr_exec_script_buffer_t* buffer,
btx_csr_exec_script_callback_fp callback, void* callback_param);
```

Description

brief Patch controller's firmware ingroup btx_csr

details This function executes a script that patches the controller's firmware. The c script must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the c callback with the first parameter TRUE. Otherwise the c callback is called with the first parameter FALSE.

If support for multiple firmware revisions is neede use btx_csr_patch_controller().

param script Array of patch scripts. param buffer A buffer for storing temporary data needed for script execution. param callback The callback function that will be called when the script has been executed. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_get_cached_temperature Function

File

csr.h

C

```
bt_bool btx_csr_get_cached_temperature(btx_csr_get_var_callback_fp callback, void* callback_param);
```

Description

brief Get chip's cached temperature ingroup btx_csr

param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_get_ps_var Function

File

csr.h

C

```
bt_bool btx_csr_get_ps_var(bt_uint ps_key, bt_uint value_word_count, btx_csr_get_ps_var_callback_fp
callback, void* callback_param);
```

Description

This is function btx_csr_get_ps_var.

btx_csr_get_ps_var_ex Function

File

csr.h

C

```
bt_bool btx_csr_get_ps_var_ex(bt_uint ps_key, bt_uint value_word_count, bt_uint store,
btx_csr_get_ps_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_get_ps_var_ex.

btx_csr_get_rssi_acl Function

File

csr.h

C

```
bt_bool btx_csr_get_rssi_acl(bt_hci_hconn_t hconn, btx_csr_get_var_callback_fp callback, void*
callback_param);
```

Description

brief Get RSSI ingroup btx_csr

details This function retrieves the RSSI for a given HCI ACL handle.

param hconn ACL connection handle. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_get_script__PB_27_R20_BC6ROM_A04 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_27_R20_BC6ROM_A04();
```

Description

brief Return script for patching BlueCore 6 ingroup btx_csr

btx_csr_get_script__PB_90_REV6 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_90_REV6();
```

Description

brief Return script for patching CSR8810 (BlueCore 7) ingroup btx_csr

btx_csr_get_var Function

File

csr.h

C

```
bt_bool btx_csr_get_var(bt_uint var_id, btx_csr_get_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_get_var.

btx_csr_set_ps_var Function

File

csr.h

C

```
bt_bool btx_csr_set_ps_var(bt_uint ps_key, const bt_uint* value, bt_uint value_word_count,
bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_csr_set_ps_var.

btx_csr_set_ps_var_ex Function

File

csr.h

C

```
bt_bool btx_csr_set_ps_var_ex(bt_uint ps_key, const bt_uint* value, bt_uint value_word_count, bt_uint
store, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_csr_set_ps_var_ex.

btx_csr_set_ps_vars Function

File

csr.h

C

```
void btx_csr_set_ps_vars(const bt_uint* ps_vars, btx_csr_set_ps_vars_buffer_t* buffer,
btx_csr_set_ps_vars_callback_fp callback, void* callback_param);
```

Description

brief Write PS variables ingroup btx_csr

details

param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_set_ps_vars_ex Function

File

csr.h

C

```
void btx_csr_set_ps_vars_ex(const bt_uint* ps_vars, btx_csr_set_ps_vars_buffer_t* buffer, bt_uint store,
btx_csr_set_ps_vars_callback_fp callback, void* callback_param);
```

Description

brief Write PS variables ingroup btx_csr

param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param store param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_set_var Function

File

csr.h

C

```
bt_bool btx_csr_set_var(bt_uint var_id, const bt_uint* value, bt_uint value_word_count,
btx_csr_set_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_set_var.

btx_csr_warm_reset Function

File

csr.h

C

```
bt_bool btx_csr_warm_reset();
```

Description

brief Warm reset ingroup btx_csr

details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.

btx_csr_warm_reset_ex Function

File

csr.h

C

```
bt_bool btx_csr_warm_reset_ex(bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Warm reset ingroup btx_csr

details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the c callback is called right after the command packet has been transmitted to the controller.

param callback The callback function that will be called after the warm reset command has been sent to the controller. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_ti_drpb_enable_rf_calibration Function

File

ti.h

C

```
bt_bool btx_ti_drpb_enable_rf_calibration(bt_byte periodic_mode, bt_ulong calibration_procedure, bt_byte
temp_condition, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_enable_rf_calibration.

btx_ti_drpb_set_power_vector Function

File

ti.h

C

```
bt_bool btx_ti_drpb_set_power_vector(bt_byte modulation_type, const bt_byte* power_vector, bt_byte
epc_max_level_threshold, bt_uint external_pa_mode, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_set_power_vector.

btx_ti_drpb_tester_con_tx Function

File

ti.h

C

```
bt_bool btx_ti_drpb_tester_con_tx(bt_byte modulation_scheme, bt_byte test_pattern, bt_byte
frequency_channel, bt_byte power_level, bt_ulong generator_initialization_value, bt_ulong
edr_generator_mask, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_tester_con_tx.

btx_ti_enable_deep_sleep Function

File

ti.h

C

```
bt_bool btx_ti_enable_deep_sleep(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_deep_sleep.

btx_ti_enable_fast_clock_crystal Function

File

ti.h

C

```
bt_bool btx_ti_enable_fast_clock_crystal(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_fast_clock_crystal.

btx_ti_enable_low_power_scan Function

File

ti.h

C

```
bt_bool btx_ti_enable_low_power_scan(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_low_power_scan.

btx_ti_enable_low_power_scan_default Function

File

ti.h

C

```
bt_bool btx_ti_enable_low_power_scan_default(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_low_power_scan_default.

btx_ti_exec_script Function

File

ti.h

C

```
void btx_ti_exec_script(const btx_ti_script_t* script, btx_ti_exec_script_buffer_t* buffer,
btx_ti_completion_callback_fp callback, void* callback_param);
```

Description

This is function btx_ti_exec_script.

btx_ti_exec_script_oem Function

File

ti.h

C

```
void btx_ti_exec_script_oem(const btx_ti_script_t* script, btx_ti_exec_script_oem_buffer_t* buffer,
btx_ti_exec_script_oem_callback_fp callback, void* callback_param);
```

Description

This is function btx_ti_exec_script_oem.

btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36();
```

Description

CC2560 Scripts (Service pack 2.36)

btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44();
```

Description

CC2560 Scripts (Service pack 2.44)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_10 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_10();
```

Description

CC2564 Scripts (Service pack 2.10)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_12 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_12();
```

Description

CC2564 Scripts (Service pack 2.12)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_7 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_7();
```

Description

CC2564 Scripts (Service pack 2.7)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short();
```

Description

CC2564 Scripts (Service pack 2.8)

btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn.

btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_0_1 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_1();
```

Description

CC2564B Scripts (Service pack 0.1)

btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_0_2 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_2();
```

Description

CC2564B Scripts (Service pack 0.2)

btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3();
```

Description

CC2564 Scripts (Service pack 2.3)

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4();
```

Description

CC2564 Scripts (Service pack 2.4)

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn.

btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3();
```

Description

CC2564 Scripts (Service pack 1.3)

btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init();
```

Description

This is function btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init.

btx_ti_init_ble_controller Function

File

ti.h

C

```
void btx_ti_init_ble_controller(btx_ti_exec_script_buffer_t* buffer, btx_ti_completion_callback_fp
callback, void* callback_param);
```

Description

This is function btx_ti_init_ble_controller.

btx_ti_init_controller Function

File

ti.h

C

```
void btx_ti_init_controller(btx_ti_exec_script_buffer_t* buffer, btx_ti_completion_callback_fp callback,
void* callback_param);
```

Description

This is function btx_ti_init_controller.

btx_ti_set_uart_baud_rate Function

File

ti.h

C

```
bt_bool btx_ti_set_uart_baud_rate(bt_ulong baud_rate, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_set_uart_baud_rate.

btx_ti_write_bdaddr Function

File

ti.h

C

```
bt_bool btx_ti_write_bdaddr(bt_bdaddr_t* bdaddr, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_write_bdaddr.

btx_ti_write_hardware_register Function

File

ti.h

C

```
bt_bool btx_ti_write_hardware_register(bt_ulong address, bt_uint value, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_write_hardware_register.

btx_csr_get_script__dsp_script__PB_109_DSP_rev8 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__dsp_script__PB_109_DSP_rev8();
```

Description

brief Return script for patching DSP in CSR8x11 A08 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_101_CSR8811_CSP28_UART Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_101_CSR8811_CSP28_UART();
```

Description

brief Return script for patching CSR8x11 A06 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_109_CSR8811_REV16 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_109_CSR8811_REV16();
```

Description

brief Return script for patching CSR8x11 A08 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_173_CSR8X11_REV1 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_173_CSR8X11_REV1();
```

Description

brief Return script for patching CSR8x11 A12 (BlueCore 7) ingroup btx_csr

btx_csr_init Function

File

csr.h

C

```
void btx_csr_init();
```

Description

brief Initialize CSR support layer. ingroup btx_csr

details This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.

btx_csr_init_hq_script Function

File

csr.h

C

```
void btx_csr_init_hq_script(const btx_csr_script_t* script, btx_csr_exec_hq_script_buffer_t* buffer);
```

Description

This is function btx_csr_init_hq_script.

btx_csr_patch_controller Function

File

csr.h

C

```
void btx_csr_patch_controller(const btx_csr_get_script_fp* scripts, bt_int script_count,
btx_csr_exec_script_buffer_t* buffer, btx_csr_exec_script_callback_fp callback, void* callback_param);
```

Description

brief Patch controller's firmware ingroup btx_csr

details This function executes a script that patches the controller's firmware. Each entry of the c scripts array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the c scripts. If there is a matching script it is loaded to the controller and c callback is called with the first parameter TRUE. If no suitable script found c callback is called with the first parameter FALSE.

param scripts Array of patch scripts. param script_count The number of scripts in c scripts. param buffer A buffer for storing temporary data needed for script execution. param callback The callback function that will be called when the script has been executed. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_register_bccmd_listener Function

File

csr.h

C

```
bt_bool btx_csr_register_bccmd_listener(btx_csr_bccmd_listener_t* listener);
```

Description

This is function btx_csr_register_bccmd_listener.

btx_csr_send_dsp_config_data Function

File

csr.h

C

```
bt_bool btx_csr_send_dsp_config_data(bt_uint total_config_blocks, bt_uint seq_no, bt_uint status, const
bt_uint* data, bt_uint data_word_count, bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_send_dsp_config_data.

btx_csr_send_next_hq_script_packet Function

File

csr.h

C

```
bt_bool btx_csr_send_next_hq_script_packet(bt_uint seq_no, btx_csr_exec_hq_script_buffer_t* buffer);
```

Description

This is function btx_csr_send_next_hq_script_packet.

btx_csr_unregister_bccmd_listener Function

File

csr.h

C

```
void btx_csr_unregister_bccmd_listener(btx_csr_bccmd_listener_t* listener);
```

Description

This is function btx_csr_unregister_bccmd_listener.

btx_ti_a3dp_sink_close_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_close_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_close_stream.

btx_ti_a3dp_sink_codec_config Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_codec_config(bt_byte channels, bt_byte sample_frequency, bt_byte channel_mode,
bt_byte blocks, bt_byte sub_bands, bt_byte allocation_method, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_codec_config.

btx_ti_a3dp_sink_open_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_open_stream(bt_byte acl_handle, bt_uint l2cap_cid, bt_hci_cmd_callback_fp
callback);
```

Description

This is function btx_ti_a3dp_sink_open_stream.

btx_ti_a3dp_sink_start_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_start_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_start_stream.

btx_ti_a3dp_sink_stop_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_stop_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_stop_stream.

btx_ti_avpr_debug Function

File

ti.h

C

```
bt_bool btx_ti_avpr_debug(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_avpr_debug.

btx_ti_avpr_enable Function

File

ti.h

C

```
bt_bool btx_ti_avpr_enable(bt_byte enable, bt_byte load_a3dp_code, bt_byte a3dp_role,
bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_avpr_enable.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_14 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_14();
```

Description

CC2564 Scripts (Service pack 2.14)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_0 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_0();
```

Description

CC2564B Scripts (Service pack 1.0)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_1 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_1();
```

Description

CC2564B Scripts (Service pack 1.1)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2();
```

Description

CC2564B Scripts (Service pack 1.2)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn.

btx_ti_le_enable Function

File

ti.h

C

```
bt_bool btx_ti_le_enable(bt_byte enable, bt_byte load_le_code, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_le_enable.

btx_ti_set_afh_mode Function

File

ti.h

C

```
bt_bool btx_ti_set_afh_mode(bt_hci_hconn_t hconn, bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_set_afh_mode.

btx_ti_write_codec_config Function

File

ti.h

C

```
bt_bool btx_ti_write_codec_config(const btx_ti_codec_config_t* codec_config, bt_hci_cmd_callback_fp
callback);
```

Description

This is function btx_ti_write_codec_config.

btx_csr_exec_hq_script Function

File

csr.h

C

```
void btx_csr_exec_hq_script(const btx_csr_script_t* script, btx_csr_exec_hq_script_buffer_t* buffer,
btx_csr_exec_hq_script_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_exec_hq_script.

Misc. Data Types and Constants

CSR_SNK_ADC Macro

File

csr.h

C

```
#define CSR_SNK_ADC 3
```

Description

This is macro CSR_SNK_ADC.

CSR_SNK_FASTPIPE Macro

File

csr.h

C

```
#define CSR_SNK_FASTPIPE 8
```

Description

This is macro CSR_SNK_FASTPIPE.

CSR_SNK_FM Macro

File

csr.h

C

```
#define CSR_SNK_FM 4
```

Description

This is macro CSR_SNK_FM.

CSR_SNK_I2S Macro

File

csr.h

C

```
#define CSR_SNK_I2S 2
```

Description

This is macro CSR_SNK_I2S.

CSR_SNK_L2CAP Macro

File

csr.h

C

```
#define CSR_SNK_L2CAP 7
```

Description

This is macro CSR_SNK_L2CAP.

CSR_SNK_PCM Macro

File

csr.h

C

```
#define CSR_SNK_PCM 1
```

Description

This is macro CSR_SNK_PCM.

CSR_SNK_SCO Macro

File

csr.h

C

```
#define CSR_SNK_SCO 9
```

Description

This is macro CSR_SNK_SCO.

CSR_SNK_SPDIF Macro

File

csr.h

C

```
#define CSR_SNK_SPDIF 5
```

Description

This is macro CSR_SNK_SPDIF.

CSR_SRC_ADC Macro

File

csr.h

C

```
#define CSR_SRC_ADC 3
```

Description

This is macro CSR_SRC_ADC.

CSR_SRC_FASTPIPE Macro

File

csr.h

C

```
#define CSR_SRC_FASTPIPE 8
```

Description

This is macro CSR_SRC_FASTPIPE.

CSR_SRC_FM Macro

File

csr.h

C

```
#define CSR_SRC_FM 4
```

Description

This is macro CSR_SRC_FM.

CSR_SRC_I2S Macro

File

csr.h

C

```
#define CSR_SRC_I2S 2
```

Description

This is macro CSR_SRC_I2S.

CSR_SRC_L2CAP Macro

File

csr.h

C

```
#define CSR_SRC_L2CAP 7
```

Description

This is macro CSR_SRC_L2CAP.

CSR_SRC_MIC Macro

File

csr.h

C

```
#define CSR_SRC_MIC 6
```

Description

This is macro CSR_SRC_MIC.

CSR_SRC_PCM Macro

File

csr.h

C

```
#define CSR_SRC_PCM 1
```

Description

This is macro CSR_SRC_PCM.

CSR_SRC_SCO Macro

File

csr.h

C

```
#define CSR_SRC_SCO 9
```

Description

This is macro CSR_SRC_SCO.

CSR_SRC_SPDIF Macro

File

csr.h

C

```
#define CSR_SRC_SPDIF 5
```

Description

This is macro CSR_SRC_SPDIF.

CSR_VARID_CACHED_TEMPERATURE Macro

File

csr.h

C

```
#define CSR_VARID_CACHED_TEMPERATURE 0x2872
```

Description

This is macro CSR_VARID_CACHED_TEMPERATURE.

CSR_VARID_ENABLE_SCO_STREAMS Macro

File

csr.h

C

```
#define CSR_VARID_ENABLE_SCO_STREAMS 0x4876
```

Description

This is macro CSR_VARID_ENABLE_SCO_STREAMS.

CSR_VARID_MAP_SCO_AUDIO Macro

File

csr.h

C

```
#define CSR_VARID_MAP_SCO_AUDIO 0x506a
```

Description

This is macro CSR_VARID_MAP_SCO_AUDIO.

CSR_VARID_PIO Macro

File

csr.h

C

```
#define CSR_VARID_PIO 0x681f
```

Description

This is macro CSR_VARID_PIO.

CSR_VARID_PIO_DIRECTION_MASK Macro

File

csr.h

C

```
#define CSR_VARID_PIO_DIRECTION_MASK 0x681e
```

Description

This is macro CSR_VARID_PIO_DIRECTION_MASK.

CSR_VARID_PIO_PROTECT_MASK Macro

File

csr.h

C

```
#define CSR_VARID_PIO_PROTECT_MASK 0x2823
```

Description

This is macro CSR_VARID_PIO_PROTECT_MASK.

CSR_VARID_RSSI_ACL Macro

File

csr.h

C

```
#define CSR_VARID_RSSI_ACL 0x301d
```

Description

This is macro CSR_VARID_RSSI_ACL.

CSR_VARID_STREAM_CLOSE_SINK Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CLOSE_SINK 0x486c
```

Description

This is macro CSR_VARID_STREAM_CLOSE_SINK.

CSR_VARID_STREAM_CLOSE_SOURCE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CLOSE_SOURCE 0x486b
```

Description

This is macro CSR_VARID_STREAM_CLOSE_SOURCE.

CSR_VARID_STREAM_CONFIGURE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CONFIGURE 0x505c
```

Description

This is macro CSR_VARID_STREAM_CONFIGURE.

CSR_VARID_STREAM_CONNECT Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CONNECT 0x505e
```

Description

This is macro CSR_VARID_STREAM_CONNECT.

CSR_VARID_STREAM_GET_SINK Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_GET_SINK 0x505b
```

Description

This is macro CSR_VARID_STREAM_GET_SINK.

CSR_VARID_STREAM_GET_SOURCE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_GET_SOURCE 0x505a
```

Description

This is macro CSR_VARID_STREAM_GET_SOURCE.

GETRESP_BAD_REQ Macro

File

csr.h

C

```
#define GETRESP_BAD_REQ 0x0004
```

Description

This is macro GETRESP_BAD_REQ.

GETRESP_ERROR Macro

File

csr.h

C

```
#define GETRESP_ERROR 0x0008
```

Description

This is macro GETRESP_ERROR.

GETRESP_NO_ACCESS Macro

File

csr.h

C

```
#define GETRESP_NO_ACCESS 0x0005
```

Description

This is macro GETRESP_NO_ACCESS.

GETRESP_NO_SUCH_VARID Macro

File

csr.h

C

```
#define GETRESP_NO_SUCH_VARID 0x0001
```

Description

This is macro GETRESP_NO_SUCH_VARID.

GETRESP_NO_VALUE Macro

File

csr.h

C

```
#define GETRESP_NO_VALUE 0x0003
```

Description

This is macro GETRESP_NO_VALUE.

GETRESP_OK Macro

File

csr.h

C

```
#define GETRESP_OK 0x0000
```

Description

BCCMD GETRESP status codes

GETRESP_PERMISSION_DENIED Macro

File

csr.h

C

```
#define GETRESP_PERMISSION_DENIED 0x0009
```

Description

This is macro GETRESP_PERMISSION_DENIED.

GETRESP_READ_ONLY Macro

File

csr.h

C

```
#define GETRESP_READ_ONLY 0x0006
```

Description

This is macro GETRESP_READ_ONLY.

GETRESP_TOO_BIG Macro

File

csr.h

C

```
#define GETRESP_TOO_BIG 0x0002
```

Description

This is macro GETRESP_TOO_BIG.

GETRESP_WRITE_ONLY Macro

File

csr.h

C

```
#define GETRESP_WRITE_ONLY 0x0007
```

Description

This is macro GETRESP_WRITE_ONLY.

H Macro

File

patch.h

C

```
#define H(ps_key, val_len, seq) \
    0x01,          /* HCI command packet indicator.*/ \
    0x00, 0xfc,    /* HCI opcode { OGF=0x3f OCF=0x00 } */ \
                   /*     OGF=0x3f implies a manufacturer-sepcific command. */ \
                   /* Parameter length. */ \
    1 + (5 + 3 + val_len) * 2, \
    0xc2,          /* CSR payload descriptor: */ \
                   /*    bit 7: Last fragment = 1 */ \
                   /*    bit 6: First fragment = 1 */ \
                   /*    bits 5-0 Channel ID = 2 (BCCMD) */ \
    W(0x0002),     /* SETREQ message */ \
    W(5+3+val_len), /* length*/ \
    W(seq),        /* sequence number */ \
    W(0x7003),     /* Variable ID of PS command */ \
    W(0x0000),     /* Status (must be 0 in SETREQ) */ \
    \
    W(ps_key),     /* PS key */ \
    W(val_len),    /* Value length in 16-bit words */ \
    W(0)           /* PS key store (0 = use default) */
```

Description

PS key store (0 = use default)

PS_DEFAULT Macro

File

csr.h

C

```
#define PS_DEFAULT 0x0000
```

Description

This is macro PS_DEFAULT.

PS_F Macro

File

csr.h

C

```
#define PS_F 0x0002
```

Description

This is macro PS_F.

PS_I Macro

File

csr.h

C

```
#define PS_I 0x0001
```

Description

This is macro PS_I.

PS_RAM Macro

File

csr.h

C

```
#define PS_RAM 0x0008
```

Description

This is macro PS_RAM.

PS_ROM Macro

File

csr.h

C

```
#define PS_ROM 0x0004
```

Description

This is macro PS_ROM.

PSKEY_ANA_FREQ Macro

File

csr.h

C

```
#define PSKEY_ANA_FREQ 0x01fe
```

Description

This is macro PSKEY_ANA_FREQ.

PSKEY_BDADDR Macro

File

csr.h

C

```
#define PSKEY_BDADDR 0x0001
```

Description

Select PS key definitions

PSKEY_CLOCK_REQUEST_ENABLE Macro

File

csr.h

C

```
#define PSKEY_CLOCK_REQUEST_ENABLE 0x0246
```

Description

This is macro PSKEY_CLOCK_REQUEST_ENABLE.

PSKEY_DEEP_SLEEP_CLEAR_RTS Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_CLEAR_RTS 0x0252
```

Description

This is macro PSKEY_DEEP_SLEEP_CLEAR_RTS.

PSKEY_DEEP_SLEEP_STATE Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_STATE 0x0229
```

Description

This is macro PSKEY_DEEP_SLEEP_STATE.

PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK 0x03c3
```

Description

This is macro PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK.

PSKEY_DEEP_SLEEP_WAKE_CTS Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_WAKE_CTS 0x023c
```

Description

This is macro PSKEY_DEEP_SLEEP_WAKE_CTS.

PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE Macro

File

csr.h

C

```
#define PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE 0x01db
```

Description

This is macro PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE.

PSKEY_DIGITAL_AUDIO_CONFIG Macro

File

csr.h

C

```
#define PSKEY_DIGITAL_AUDIO_CONFIG 0x01d9
```

Description

This is macro PSKEY_DIGITAL_AUDIO_CONFIG.

PSKEY_HCI_NOP_DISABLE Macro

File

csr.h

C

```
#define PSKEY_HCI_NOP_DISABLE 0x00f2
```

Description

This is macro PSKEY_HCI_NOP_DISABLE.

PSKEY_HOST_INTERFACE Macro

File

csr.h

C

```
#define PSKEY_HOST_INTERFACE 0x01f9
```

Description

This is macro PSKEY_HOST_INTERFACE.

PSKEY_HOSTIO_MAP_SCO_PCM Macro

File

csr.h

C

```
#define PSKEY_HOSTIO_MAP_SCO_PCM 0x01ab
```

Description

This is macro PSKEY_HOSTIO_MAP_SCO_PCM.

PSKEY_HOSTIO_UART_RESET_TIMEOUT Macro

File

csr.h

C

```
#define PSKEY_HOSTIO_UART_RESET_TIMEOUT 0x01a4
```

Description

This is macro PSKEY_HOSTIO_UART_RESET_TIMEOUT.

PSKEY_PCM_PULL_CONTROL Macro

File

csr.h

C

```
#define PSKEY_PCM_PULL_CONTROL 0x01e2
```

Description

This is macro PSKEY_PCM_PULL_CONTROL.

PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL Macro

File

csr.h

C

```
#define PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL 0x21bd
```

Description

This is macro PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL.

PSKEY_UART_BAUDRATE Macro

File

csr.h

C

```
#define PSKEY_UART_BAUDRATE 0x01be
```

Description

This is macro PSKEY_UART_BAUDRATE.

PSKEY_UART_BITRATE Macro

File

csr.h

C

```
#define PSKEY_UART_BITRATE 0x01ea
```

Description

This is macro PSKEY_UART_BITRATE.

PSKEY_UART_CONFIG_BCSP Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_BCSP 0x01bf
```

Description

This is macro PSKEY_UART_CONFIG_BCSP.

PSKEY_UART_CONFIG_H4 Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H4 0x01c0
```

Description

This is macro PSKEY_UART_CONFIG_H4.

PSKEY_UART_CONFIG_H5 Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H5 0x01c1
```

Description

This is macro PSKEY_UART_CONFIG_H5.

PSKEY_UART_TX_WINDOW_SIZE Macro

File

csr.h

C

```
#define PSKEY_UART_TX_WINDOW_SIZE 0x01c6
```

Description

This is macro PSKEY_UART_TX_WINDOW_SIZE.

PSKEY_VM_DISABLE Macro

File

csr.h

C

```
#define PSKEY_VM_DISABLE 0x025d
```

Description

This is macro PSKEY_VM_DISABLE.

SET_PS_VALUE_BDADDR Macro

File

csr.h

C

```
#define SET_PS_VALUE_BDADDR(key, m, l) key, 4, (uint16_t)((((uint32_t)l) >> 16) & 0xFF), (uint16_t)(l
&0xFFFF), (uint16_t)((((uint32_t)l) >> 24) & 0xFF), (uint16_t)(m &0xFFFF)
```

Description

This is macro SET_PS_VALUE_BDADDR.

SET_PS_VALUE_UINT16 Macro

File

csr.h

C

```
#define SET_PS_VALUE_UINT16(key, value) key, 1, value
```

Description

Macros for defining lists of PS values for use with btx_csr_set_ps_vars().

SET_PS_VALUE_UINT32 Macro

File

csr.h

C

```
#define SET_PS_VALUE_UINT32(key, value) key, 2, (uint16_t)((((uint32_t)value) >> 16) & 0xFFFF),
(uint16_t)(value &0xFFFF)
```

Description

This is macro SET_PS_VALUE_UINT32.

W Macro

File

patch.h

C

```
#define W(w) _W((bt_uint)(w))
```

Description

This is macro W.

BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE Macro

File

ti.h

C

```
#define BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE 10
```

Description

This is macro BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE.

btx_ti_get_script__CC2560_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2560_ServicePack btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44
```

Description

Alias for the latest script

btx_ti_get_script__CC2564_BLE_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564_BLE_Init btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn
```

Description

This is macro btx_ti_get_script__CC2564_BLE_Init.

btx_ti_get_script__CC2564_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564_ServicePack btx_ti_get_script__BL6450L_BT_Service_Pack_2_14
```

Description

Aliases for latest scripts

btx_ti_get_script__CC2564B_BLE_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_BLE_Init btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn
```

Description

This is macro btx_ti_get_script__CC2564B_BLE_Init.

btx_ti_get_script__CC2564B_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_ServicePack btx_ti_get_script__CC2564B_BT_Service_Pack_1_2
```

Description

Aliases for latest scripts

BTX_TI_MODULATION_SCHEME_8_DPSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_8_DPSK 3
```

Description

This is macro BTX_TI_MODULATION_SCHEME_8_DPSK.

BTX_TI_MODULATION_SCHEME_CW Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_CW 0
```

Description

This is macro BTX_TI_MODULATION_SCHEME_CW.

BTX_TI_MODULATION_SCHEME_GFSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_GFSK 1
```

Description

This is macro BTX_TI_MODULATION_SCHEME_GFSK.

BTX_TI_MODULATION_SCHEME_P4_DPSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_P4_DPSK 2
```

Description

This is macro BTX_TI_MODULATION_SCHEME_P4_DPSK.

BTX_TI_MODULATION_TYPE_EDR2 Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_EDR2 1
```

Description

This is macro BTX_TI_MODULATION_TYPE_EDR2.

BTX_TI_MODULATION_TYPE_EDR3 Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_EDR3 2
```

Description

This is macro BTX_TI_MODULATION_TYPE_EDR3.

BTX_TI_MODULATION_TYPE_GFSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_GFSK 0
```

Description

This is macro BTX_TI_MODULATION_TYPE_GFSK.

BTX_TI_TEST_PATTERN_ALL_0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_ALL_0 4
```

Description

This is macro BTX_TI_TEST_PATTERN_ALL_0.

BTX_TI_TEST_PATTERN_ALL_1 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_ALL_1 3
```

Description

This is macro BTX_TI_TEST_PATTERN_ALL_1.

BTX_TI_TEST_PATTERN_F0F0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_F0F0 5
```

Description

This is macro BTX_TI_TEST_PATTERN_F0F0.

BTX_TI_TEST_PATTERN_FF00 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_FF00 6
```

Description

This is macro BTX_TI_TEST_PATTERN_FF00.

BTX_TI_TEST_PATTERN_PN15 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_PN15 1
```

Description

This is macro BTX_TI_TEST_PATTERN_PN15.

BTX_TI_TEST_PATTERN_PN9 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_PN9 0
```

Description

This is macro BTX_TI_TEST_PATTERN_PN9.

BTX_TI_TEST_PATTERN_USER Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_USER 7
```

Description

This is macro BTX_TI_TEST_PATTERN_USER.

BTX_TI_TEST_PATTERN_Z0Z0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_Z0Z0 2
```

Description

This is macro BTX_TI_TEST_PATTERN_Z0Z0.

_W Macro

File

patch.h

C

```
#define _W(w) (bt_byte)((w) & 0xFF), (bt_byte)((w) >> 8)
```

Description

This is macro _W.

_btx_csr_autobaud_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_autobaud_buffer_t {
  bt_byte* recv_buffer;
  bt_uint recv_buffer_len;
  btx_csr_autobaud_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_autobaud_buffer_t.

_btx_csr_exec_script_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_exec_script_buffer_t {
  const btx_csr_script_t* script;
  btx_csr_exec_script_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
};
```

Description

This is record _btx_csr_exec_script_buffer_t.

_btx_csr_set_ps_vars_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_set_ps_vars_buffer_t {
  const bt_uint* ps_vars;
  btx_csr_set_ps_vars_callback_fp callback;
  void* callback_param;
  bt_uint current_var;
};
```

Description

This is record _btx_csr_set_ps_vars_buffer_t.

_btx_ti_exec_script_buffer_t Structure

File

ti.h

C

```
struct _btx_ti_exec_script_buffer_t {
  const btx_ti_script_t* script;
  btx_ti_completion_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
};
```

Description

This is record _btx_ti_exec_script_buffer_t.

_btx_ti_exec_script_oem_buffer_t Structure

File

ti.h

C

```
struct _btx_ti_exec_script_oem_buffer_t {
  const btx_ti_script_t* script;
  btx_ti_exec_script_oem_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
  bt_int state;
  bt_byte rx_buffer[BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE];
};
```

Description

This is record _btx_ti_exec_script_oem_buffer_t.

btx_csr_autobaud_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_autobaud_buffer_t btx_csr_autobaud_buffer_t;
```

Description

This is type btx_csr_autobaud_buffer_t.

btx_csr_autobaud_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_autobaud_callback_fp)(bt_bool success, btx_csr_autobaud_buffer_t* buffer);
```

Description

This is type btx_csr_autobaud_callback_fp.

btx_csr_bccmd_header_t Structure

File

csr.h

C

```
typedef struct _btx_csr_bccmd_header_s {
  bt_uint type;
  bt_uint len;
  bt_uint seq_no;
  bt_uint var_id;
  bt_uint status;
  bt_byte* payload;
} btx_csr_bccmd_header_t;
```

Description

This is type btx_csr_bccmd_header_t.

btx_csr_cached_temperature_t Structure

File

csr.h

C

```
typedef struct _btx_csr_cached_temperature_s {
  btx_csr_bccmd_header_t message;
  bt_uint temperature;
} btx_csr_cached_temperature_t;
```

Description

This is type btx_csr_cached_temperature_t.

btx_csr_exec_script_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_exec_script_buffer_t btx_csr_exec_script_buffer_t;
```

Description

This is type btx_csr_exec_script_buffer_t.

btx_csr_exec_script_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_exec_script_callback_fp)(bt_bool success, btx_csr_exec_script_buffer_t* buffer);
```

Description

This is type btx_csr_exec_script_callback_fp.

btx_csr_get_ps_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_get_ps_var_callback_fp)(bt_uint success, const bt_byte* value, bt_uint len, void*
callback_param);
```

Description

This is type btx_csr_get_ps_var_callback_fp.

btx_csr_get_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_get_var_callback_fp)(bt_uint status, bt_uint var_id, btx_csr_var_t* var_value,
void* callback_param);
```

Description

This is type btx_csr_get_var_callback_fp.

btx_csr_pio_direction_mask_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_direction_mask_s {
  btx_csr_bccmd_header_t message;
  bt_uint direction;
} btx_csr_pio_direction_mask_t;
```

Description

This is type btx_csr_pio_direction_mask_t.

btx_csr_pio_protection_mask_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_protection_mask_s {
  btx_csr_bccmd_header_t message;
  bt_uint protection;
} btx_csr_pio_protection_mask_t;
```

Description

This is type btx_csr_pio_protection_mask_t.

btx_csr_pio_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_s {
  btx_csr_bccmd_header_t message;
  bt_uint pio;
} btx_csr_pio_t;
```

Description

This is type btx_csr_pio_t.

btx_csr_rssi_acl_t Structure

File

csr.h

C

```
typedef struct _btx_csr_rssi_acl_s {
  btx_csr_bccmd_header_t message;
  bt_hci_hconn_t hconn;
  bt_uint rssi;
} btx_csr_rssi_acl_t;
```

Description

This is type btx_csr_rssi_acl_t.

btx_csr_script_t Structure

File

csr.h

C

```
typedef struct _btx_csr_script_t {
  const bt_byte* const * packets;
  bt_int packet_count;
  bt_uint revision;
} btx_csr_script_t;
```

Description

This is type btx_csr_script_t.

btx_csr_set_ps_vars_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_set_ps_vars_buffer_t btx_csr_set_ps_vars_buffer_t;
```

Description

This is type btx_csr_set_ps_vars_buffer_t.

btx_csr_set_ps_vars_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_set_ps_vars_callback_fp)(bt_bool success, btx_csr_set_ps_vars_buffer_t* buffer);
```

Description

This is type btx_csr_set_ps_vars_callback_fp.

btx_csr_set_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_set_var_callback_fp)(bt_uint status, bt_uint var_id, btx_csr_var_t* var_value,
void* callback_param);
```

Description

This is type btx_csr_set_var_callback_fp.

btx_csr_strm_get_sink_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_get_sink_s {
  btx_csr_bccmd_header_t message;
  bt_uint sink_id;
} btx_csr_strm_get_sink_t;
```

Description

This is type btx_csr_strm_get_sink_t.

btx_csr_strm_get_source_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_get_source_s {
  btx_csr_bccmd_header_t message;
  bt_uint source_id;
} btx_csr_strm_get_source_t;
```

Description

This is type btx_csr_strm_get_source_t.

btx_csr_var_t Union

File

csr.h

C

```
typedef union _btx_csr_var_u {
  btx_csr_bccmd_header_t message;
  btx_csr_cached_temperature_t cached_temperature;
  btx_csr_rssi_acl_t rssi_acl;
  btx_csr_pio_t pio;
  btx_csr_pio_direction_mask_t pio_direction;
  btx_csr_pio_protection_mask_t pio_protection;
  btx_csr_strm_get_sink_t strm_get_sink;
  btx_csr_strm_get_source_t strm_get_source;
  btx_csr_create_operator_c_t create_operator;
  btx_csr_strm_connect_t strm_connect;
} btx_csr_var_t;
```

Description

This is type btx_csr_var_t.

btx_ti_completion_callback_fp Type

File

ti.h

C

```
typedef void (* btx_ti_completion_callback_fp)(bt_bool success, btx_ti_exec_script_buffer_t* buffer, void*
callback_param);
```

Description

This is type btx_ti_completion_callback_fp.

btx_ti_exec_script_buffer_t Type

File

ti.h

C

```
typedef struct _btx_ti_exec_script_buffer_t btx_ti_exec_script_buffer_t;
```

Description

This is type btx_ti_exec_script_buffer_t.

btx_ti_exec_script_oem_buffer_t Type

File

ti.h

C

```
typedef struct _btx_ti_exec_script_oem_buffer_t btx_ti_exec_script_oem_buffer_t;
```

Description

This is type btx_ti_exec_script_oem_buffer_t.

btx_ti_exec_script_oem_callback_fp Type

File

ti.h

C

```
typedef void (* btx_ti_exec_script_oem_callback_fp)(bt_bool success, btx_ti_exec_script_oem_buffer_t*
buffer, void* callback_param);
```

Description

This is type btx_ti_exec_script_oem_callback_fp.

btx_ti_script_t Structure

File

ti.h

C

```
typedef struct _btx_ti_script_t {
  const bt_byte* const * packets;
  bt_int packet_count;
  bt_byte fw_version_x;
  bt_byte fw_version_z;
} btx_ti_script_t;
```

Description

This is type btx_ti_script_t.

BTX_TI_A3DP_ROLE_SINK Macro

File

ti.h

C

```
#define BTX_TI_A3DP_ROLE_SINK 1
```

Description

This is macro BTX_TI_A3DP_ROLE_SINK.

BTX_TI_A3DP_ROLE_SOURCE Macro

File

ti.h

C

```
#define BTX_TI_A3DP_ROLE_SOURCE 0
```

Description

This is macro BTX_TI_A3DP_ROLE_SOURCE.

BTX_TI_AVPR_DISABLE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_DISABLE 0
```

Description

This is macro BTX_TI_AVPR_DISABLE.

BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE 0
```

Description

This is macro BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE.

BTX_TI_AVPR_ENABLE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_ENABLE 1
```

Description

This is macro BTX_TI_AVPR_ENABLE.

BTX_TI_AVPR_LOAD_A3DP_CODE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_LOAD_A3DP_CODE 1
```

Description

This is macro BTX_TI_AVPR_LOAD_A3DP_CODE.

btx_ti_get_script__CC2564B_AVPR_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_AVPR_Init btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn
```

Description

This is macro btx_ti_get_script__CC2564B_AVPR_Init.

BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS Macro

File

ti.h

C

```
#define BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS 0
```

Description

This is macro BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS.

BTX_TI_SBC_ALLOCATION_METHOD_SNR Macro

File

ti.h

C

```
#define BTX_TI_SBC_ALLOCATION_METHOD_SNR 1
```

Description

This is macro BTX_TI_SBC_ALLOCATION_METHOD_SNR.

BTX_TI_SBC_BLOCKS_12 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_12 12
```

Description

This is macro BTX_TI_SBC_BLOCKS_12.

BTX_TI_SBC_BLOCKS_16 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_16 16
```

Description

This is macro BTX_TI_SBC_BLOCKS_16.

BTX_TI_SBC_BLOCKS_4 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_4 4
```

Description

This is macro BTX_TI_SBC_BLOCKS_4.

BTX_TI_SBC_BLOCKS_8 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_8 8
```

Description

This is macro BTX_TI_SBC_BLOCKS_8.

BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL 1
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL.

BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO 3
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO.

BTX_TI_SBC_CHANNEL_MODE_MONO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_MONO 0
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_MONO.

BTX_TI_SBC_CHANNEL_MODE_STEREO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_STEREO 2
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_STEREO.

BTX_TI_SBC_SAMPLE_FREQUENCY_16000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_16000 0
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_16000.

BTX_TI_SBC_SAMPLE_FREQUENCY_32000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_32000 1
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_32000.

BTX_TI_SBC_SAMPLE_FREQUENCY_44100 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_44100 2
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_44100.

BTX_TI_SBC_SAMPLE_FREQUENCY_48000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_48000 3
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_48000.

CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN 0x207
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN.

CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE 0x206
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE.

CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE 0x20A
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE.

CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE 0x209
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY 0x204
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT 0x203
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION 0x208
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION.

CSR_I2S_STREAM_CFG_KEY_MASTER Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_MASTER 0x202
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_MASTER.

CSR_I2S_STREAM_CFG_KEY_MCLK Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_MCLK 0x201
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_MCLK.

CSR_I2S_STREAM_CFG_KEY_POLARITY Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_POLARITY 0x205
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_POLARITY.

CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE 0x20C
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE.

CSR_I2S_STREAM_CFG_KEY_SYNC_RATE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_SYNC_RATE 0x200
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_SYNC_RATE.

CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE 0x20B
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE.

CSR_MAX_HQ_PACKET_LEN Macro

File

csr.h

C

```
#define CSR_MAX_HQ_PACKET_LEN 129
```

Description

This is macro CSR_MAX_HQ_PACKET_LEN.

CSR_MSG_TYPE_GETREQ Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_GETREQ 0
```

Description

This is macro CSR_MSG_TYPE_GETREQ.

CSR_MSG_TYPE_GETRESP Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_GETRESP 1
```

Description

This is macro CSR_MSG_TYPE_GETRESP.

CSR_MSG_TYPE_SETREQ Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_SETREQ 2
```

Description

This is macro CSR_MSG_TYPE_SETREQ.

CSR_PCM_STREAM_CFG_KEY_LSB_FIRST Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_LSB_FIRST 0x108
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_LSB_FIRST.

CSR_PCM_STREAM_CFG_KEY_MANCHESTER Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MANCHESTER 0x104
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER.

CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE 0x106
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE.

CSR_PCM_STREAM_CFG_KEY_MASTER Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MASTER 0x102
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MASTER.

CSR_PCM_STREAM_CFG_KEY_MCLK Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MCLK 0x101
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MCLK.

CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC 0x105
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC.

CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND 0x107
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND.

CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT 0x103
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT.

CSR_PCM_STREAM_CFG_KEY_SYNC_RATE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SYNC_RATE 0x100
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SYNC_RATE.

CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE 0x109
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE.

CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION Macro

File

csr.h

C

```
#define CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION 0x101b
```

Description

This is macro CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION.

CSR_VARID_CREATE_OPERATOR_C Macro

File

csr.h

C

```
#define CSR_VARID_CREATE_OPERATOR_C 0x5075
```

Description

This is macro CSR_VARID_CREATE_OPERATOR_C.

CSR_VARID_CREATE_OPERATOR_P Macro

File

csr.h

C

```
#define CSR_VARID_CREATE_OPERATOR_P 0x5076
```

Description

This is macro CSR_VARID_CREATE_OPERATOR_P.

CSR_VARID_DESTROY_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_DESTROY_OPERATORS 0x5073
```

Description

This is macro CSR_VARID_DESTROY_OPERATORS.

CSR_VARID_DSPMANAGER_CONFIG_REQUEST Macro

File

csr.h

C

```
#define CSR_VARID_DSPMANAGER_CONFIG_REQUEST 0x101c
```

Description

This is macro CSR_VARID_DSPMANAGER_CONFIG_REQUEST.

CSR_VARID_MESSAGE_FROM_OPERATOR Macro

File

csr.h

C

```
#define CSR_VARID_MESSAGE_FROM_OPERATOR 0x101a
```

Description

Select Variable definitions

CSR_VARID_START_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_START_OPERATORS 0x5070
```

Description

This is macro CSR_VARID_START_OPERATORS.

CSR_VARID_STOP_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_STOP_OPERATORS 0x5071
```

Description

This is macro CSR_VARID_STOP_OPERATORS.

CSR_VARID_STREAM_DRAINED_NOTIFICATION Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_DRAINED_NOTIFICATION 0x101d
```

Description

This is macro CSR_VARID_STREAM_DRAINED_NOTIFICATION.

CSR_VARID_STREAM_TRANSFORM_DISCONNECT Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_TRANSFORM_DISCONNECT 0x486d
```

Description

This is macro CSR_VARID_STREAM_TRANSFORM_DISCONNECT.

PSKEY_BLE_DEFAULT_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_BLE_DEFAULT_TX_POWER 0x22c8
```

Description

This is macro PSKEY_BLE_DEFAULT_TX_POWER.

PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO 0x2579
```

Description

This is macro PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO.

PSKEY_H_HC_FC_MAX_SCO_PKT_LEN Macro

File

csr.h

C

```
#define PSKEY_H_HC_FC_MAX_SCO_PKT_LEN 0x0012
```

Description

This is macro PSKEY_H_HC_FC_MAX_SCO_PKT_LEN.

PSKEY_H_HC_FC_MAX_SCO_PKTS Macro

File

csr.h

C

```
#define PSKEY_H_HC_FC_MAX_SCO_PKTS 0x0014
```

Description

This is macro PSKEY_H_HC_FC_MAX_SCO_PKTS.

PSKEY_PCM_CLOCK_RATE Macro

File

csr.h

C

```
#define PSKEY_PCM_CLOCK_RATE 0x23bc
```

Description

This is macro PSKEY_PCM_CLOCK_RATE.

PSKEY_PCM_CONFIG32 Macro

File

csr.h

C

```
#define PSKEY_PCM_CONFIG32 0x01b3
```

Description

This is macro PSKEY_PCM_CONFIG32.

PSKEY_PCM_FORMAT Macro

File

csr.h

C

```
#define PSKEY_PCM_FORMAT 0x01b6
```

Description

This is macro PSKEY_PCM_FORMAT.

PSKEY_PCM_SYNC_RATE Macro

File

csr.h

C

```
#define PSKEY_PCM_SYNC_RATE 0x23c3
```

Description

This is macro PSKEY_PCM_SYNC_RATE.

PSKEY_PCM_USE_LOW_JITTER_MODE Macro

File

csr.h

C

```
#define PSKEY_PCM_USE_LOW_JITTER_MODE 0x23c9
```

Description

This is macro PSKEY_PCM_USE_LOW_JITTER_MODE.

PSKEY_UART_CONFIG_H4DS Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H4DS 0x01cb
```

Description

This is macro PSKEY_UART_CONFIG_H4DS.

BT_LOG_LEVEL_A2DP_PAKCET Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_A2DP_PAKCET 10
```

Description

This is macro BT_LOG_LEVEL_A2DP_PAKCET.

DCRH Macro

File

patch.h

C

```
#define DCRH(block_count, val_len, seq) \
    0x01,         /* HCI command packet indicator.*/ \
    0x00, 0xfc,   /* HCI opcode { OGF=0x3f OCF=0x00 } */ \
    /*     OGF=0x3f implies a manufacturer-sepcific command. */ \
    /* Parameter length. */ \
    1 + (5 + 1 + val_len) * 2, \
    0xc3,         /* CSR payload descriptor: */ \
    /*    bit 7: Last fragment = 1 */ \
    /*    bit 6: First fragment = 1 */ \
    /*    bits 5-0 Channel ID = 2 (BCCMD) */ \
    W(0x0001),    /* GETRESP message */ \
    W(5+1+val_len), /* length*/ \
    W(seq),       /* sequence number */ \
    W(0x101c),    /* Variable ID of DSP manager config request */ \
    W(0x0000),    /* Status (must be 0 in SETREQ) */ \
    \
    W(block_count)     /* Number of configuration blocks */
```

Description

Number of configuration blocks

btx_csr_bccmd_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_bccmd_callback_fp)(btx_csr_bccmd_header_t* message, void* cb_param);
```

Description

This is type btx_csr_bccmd_callback_fp.

btx_csr_bccmd_listener_t Type

File

csr.h

C

```
typedef struct _btx_csr_bccmd_listener_t btx_csr_bccmd_listener_t;
```

Description

This is type btx_csr_bccmd_listener_t.

btx_csr_create_operator_c_t Structure

File

csr.h

C

```
typedef struct _btx_csr_create_operator_c_s {
  btx_csr_bccmd_header_t message;
  bt_uint operator_id;
  bt_uint item_count;
  bt_uint skip_count;
  bt_uint skip_flag;
} btx_csr_create_operator_c_t;
```

Description

This is type btx_csr_create_operator_c_t.

btx_csr_exec_hq_script_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_exec_hq_script_buffer_t btx_csr_exec_hq_script_buffer_t;
```

Description

This is type btx_csr_exec_hq_script_buffer_t.

btx_csr_exec_hq_script_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_exec_hq_script_callback_fp)(bt_bool success, btx_csr_exec_hq_script_buffer_t*
buffer);
```

Description

This is type btx_csr_exec_hq_script_callback_fp.

btx_csr_get_script_fp Type

File

csr.h

C

```
typedef const btx_csr_script_t* (* btx_csr_get_script_fp)(void);
```

Description

This is type btx_csr_get_script_fp.

btx_csr_strm_connect_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_connect_s {
  btx_csr_bccmd_header_t message;
  bt_uint transform_id;
} btx_csr_strm_connect_t;
```

Description

This is type btx_csr_strm_connect_t.

btx_ti_codec_config_t Structure

File

ti.h

C

```
typedef struct _btx_ti_codec_config_s {
  bt_uint clock_rate;
  bt_byte clock_direction;
  bt_ulong frame_sync_freq;
  bt_uint frame_sync_duty_cycle;
  bt_byte frame_sync_edge;
  bt_byte frame_sync_polarity;
  bt_byte frame_sync_multiplier;
  bt_uint ch1_data_out_size;
  bt_uint ch1_data_out_offset;
  bt_byte ch1_data_out_edge;
  bt_uint ch1_data_in_size;
  bt_uint ch1_data_in_offset;
  bt_byte ch1_data_in_edge;
  bt_uint ch2_data_out_size;
  bt_uint ch2_data_out_offset;
  bt_byte ch2_data_out_edge;
  bt_uint ch2_data_in_size;
  bt_uint ch2_data_in_offset;
  bt_byte ch2_data_in_edge;
} btx_ti_codec_config_t;
```

Description

This is type btx_ti_codec_config_t.

_btx_csr_bccmd_listener_t Structure

File

csr.h

C

```
struct _btx_csr_bccmd_listener_t {
  btx_csr_bccmd_listener_t* next_listener;
  btx_csr_bccmd_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_bccmd_listener_t.

_btx_csr_exec_hq_script_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_exec_hq_script_buffer_t {
  const btx_csr_script_t* script;
  bt_int current_packet;
  bt_byte packet[CSR_MAX_HQ_PACKET_LEN];
  bt_bool success;
  btx_csr_exec_hq_script_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_exec_hq_script_buffer_t.

PSKEY_LC_DEFAULT_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_LC_DEFAULT_TX_POWER 0x0021
```

Description

This is macro PSKEY_LC_DEFAULT_TX_POWER.

PSKEY_LC_MAX_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_LC_MAX_TX_POWER 0x0017
```

Description

This is macro PSKEY_LC_MAX_TX_POWER.

PSKEY_LC_MAX_TX_POWER_NO_RSSI Macro

File

csr.h

C

```
#define PSKEY_LC_MAX_TX_POWER_NO_RSSI 0x002d
```

Description

This is macro PSKEY_LC_MAX_TX_POWER_NO_RSSI.

Utility Functions

bt_alloc_buffer Function

File

buffer.h

C

```
void* bt_alloc_buffer(bt_buffer_mgr_p pmgr);
```

Description

This is function bt_alloc_buffer.

bt_at_parse_fragment Function

File

at_parser.h

C

```
void bt_at_parse_fragment(bt_at_parser_t* parser, bt_byte* data, bt_uint len);
```

Description

This is function bt_at_parse_fragment.

bt_at_parser_reset Function

File

at_parser.h

C

```
void bt_at_parser_reset(bt_at_parser_t* parser);
```

Description

This is function bt_at_parser_reset.

bt_free_buffer Function

File

buffer.h

C

```
void bt_free_buffer(bt_buffer_mgr_p pmgr, void* data);
```

Description

This is function bt_free_buffer.

bt_get_buffer Function

File

buffer.h

C

```
void* bt_get_buffer(bt_buffer_mgr_p pmgr, bt_int index);
```

Description

This is function bt_get_buffer.

bt_get_buffer_header Function

File

buffer.h

C

```
bt_buffer_header_t* bt_get_buffer_header(bt_buffer_mgr_p pmgr, bt_int index);
```

Description

This is function bt_get_buffer_header.

bt_get_buffer_index Function

File

buffer.h

C

```
bt_int bt_get_buffer_index(bt_buffer_mgr_p pmgr, void* data);
```

Description

This is function bt_get_buffer_index.

bt_init_buffer_mgr Function

File

buffer.h

C

```
bt_bool bt_init_buffer_mgr(bt_buffer_mgr_p pmgr, bt_int num_buffers, bt_int buffer_size,
bt_buffer_header_t* buffer_headers, void* buffer_data);
```

Description

This is function bt_init_buffer_mgr.

bt_q_add Function

File

queue.h

C

```
void bt_q_add(bt_queue_element_t** phead, void* element);
```

Description

This is function bt_q_add.

bt_q_get Function

File

queue.h

C

```
void* bt_q_get(bt_queue_element_t* head, bt_int index);
```

Description

This is function bt_q_get.

bt_q_get_head Function

File

queue.h

C

```
void* bt_q_get_head(bt_queue_element_t** phead, bt_bool remove);
```

Description

This is function bt_q_get_head.

bt_q_get_length Function

File

queue.h

C

```
bt_int bt_q_get_length(bt_queue_element_t* phead);
```

Description

This is function bt_q_get_length.

bt_q_get_next Function

File

queue.h

C

```
void* bt_q_get_next(void* element);
```

Description

This is function bt_q_get_next.

bt_q_push Function

File

queue.h

C

```
void bt_q_push(bt_queue_element_t** phead, void* element);
```

Description

This is function bt_q_push.

bt_q_remove Function

File

queue.h

C

```
void* bt_q_remove(bt_queue_element_t** phead, void* pdata);
```

Description

This is function bt_q_remove.

bt_q_remove_by_idx Function

File

queue.h

C

```
void* bt_q_remove_by_idx(bt_queue_element_t* head, bt_int index);
```

Description

This is function bt_q_remove_by_idx.

bt_vcard_parse_fragment Function

File

vcard_parser.h

C

```
void bt_vcard_parse_fragment(bt_vcard_parser_t* parser, bt_byte* data, bt_uint len);
```

Description

This is function bt_vcard_parse_fragment.

bt_vcard_parser_reset Function

File

vcard_parser.h

C

```
void bt_vcard_parser_reset(bt_vcard_parser_t* parser);
```

Description

This is function bt_vcard_parser_reset.

bt_xml_parse_fragment Function

File

xml_parser.h

C

```
void bt_xml_parse_fragment(bt_xml_parser_t* parser, bt_byte_cp data, bt_uint len);
```

Description

This is function bt_xml_parse_fragment.

bt_xml_parser_reset Function

File

xml_parser.h

C

```
void bt_xml_parser_reset(bt_xml_parser_t* parser);
```

Description

This is function bt_xml_parser_reset.

_compact_buffer Function

File

bufferutils.h

C

```
void _compact_buffer(bt_byte_p pdst, bt_uint offset, bt_byte_cp psrc, bt_uint len);
```

Description

This is function _compact_buffer.

_expand_buffer Function

File

bufferutils.h

C

```
void _expand_buffer(bt_byte_p pdst, bt_byte_cp psrc, bt_uint len, bt_uint offset);
```

Description

This is function _expand_buffer.

_read_bdaddr Function

File

bufferutils.h

C

```
void _read_bdaddr(bt_bdaddr_p pbdaddr, bt_byte_cp buffer);
```

Description

This is function _read_bdaddr.

_readb Function

File

bufferutils.h

C

```
bt_bool _readb(bt_byte_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readb.

_readi Function

File

bufferutils.h

C

```
bt_bool _readi(bt_int_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readi.

_readin Function

File

bufferutils.h

C

```
bt_bool _readin(bt_int_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readin.

_readl Function

File

bufferutils.h

C

```
bt_bool _readl(bt_long_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readl.

_readln Function

File

bufferutils.h

C

```
bt_bool _readln(bt_long_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readln.

_readui Function

File

bufferutils.h

C

```
bt_bool _readui(bt_uint_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readui.

_readuin Function

File

bufferutils.h

C

```
bt_bool _readuin(bt_uint_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readuin.

_readul Function

File

bufferutils.h

C

```
bt_bool _readul(bt_ulong_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readul.

_readuln Function

File

bufferutils.h

C

```
bt_bool _readuln(bt_ulong_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readuln.

_str2ulong_dec Function

File

bufferutils.h

C

```
bt_ulong _str2ulong_dec(const char* s);
```

Description

This is function _str2ulong_dec.

_to_lower_case Function

File

bufferutils.h

C

```
void _to_lower_case(bt_byte* str, bt_uint len);
```

Description

This is function _to_lower_case.

_ulong2str Function

File

bufferutils.h

C

```
const char* _ulong2str(bt_ulong data);
```

Description

This is function _ulong2str.

_ulong2str_dec Function

File

bufferutils.h

C

```
const char* _ulong2str_dec(bt_ulong data);
```

Description

This is function _ulong2str_dec.

_write_bdaddr Function

File

bufferutils.h

C

```
void _write_bdaddr(bt_bdaddr_cp pbdaddr, bt_byte_p buffer);
```

Description

This is function _write_bdaddr.

_writeb Function

File

bufferutils.h

C

```
bt_bool _writeb(bt_byte value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writeb.

_writei Function

File

bufferutils.h

C

```
bt_bool _writei(bt_int value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writei.

_writein Function

File

bufferutils.h

C

```
bt_bool _writein(bt_int value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writein.

_writel Function

File

bufferutils.h

C

```
bt_bool _writel(bt_long value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writel.

_writeln Function

File

bufferutils.h

C

```
bt_bool _writeln(bt_long value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writeln.

_writes Function

File

bufferutils.h

C

```
bt_bool _writes(const char* pstr, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writes.

_writesx Function

File

bufferutils.h

C

```
bt_bool _writesx(const char* pstr, bt_int str_len, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writesx.

_zero_memory Function

File

bufferutils.h

C

```
void _zero_memory(void* ptr, size_t len);
```

Description

This is function _zero_memory.

_bt_memcpy Function

File

bufferutils.h

C

```
void _bt_memcpy(bt_byte_p pdst, bt_uint dst_offset, bt_byte_cp psrc, bt_uint len, bt_uint offset);
```

Description

This is function _bt_memcpy.

_is_empty_str Function

File

bufferutils.h

C

```
bt_bool _is_empty_str(const char* s);
```

Description

This is function _is_empty_str.

_str2ulong Function

File

bufferutils.h

C

```
bt_ulong _str2ulong(const char* s);
```

Description

This is function _str2ulong.

Utility Data Types and Constants

AT_EVT_CMD_CODE Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_CODE 3
```

Description

This is macro AT_EVT_CMD_CODE.

AT_EVT_CMD_COMPLETED Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_COMPLETED 6
```

Description

This is macro AT_EVT_CMD_COMPLETED.

AT_EVT_CMD_PARAM Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_PARAM 5
```

Description

This is macro AT_EVT_CMD_PARAM.

AT_EVT_CMD_READ_CODE Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_READ_CODE 4
```

Description

This is macro AT_EVT_CMD_READ_CODE.

AT_EVT_ERROR Macro

File

at_parser.h

C

```
#define AT_EVT_ERROR 2
```

Description

This is macro AT_EVT_ERROR.

AT_EVT_OK Macro

File

at_parser.h

C

```
#define AT_EVT_OK 1
```

Description

This is macro AT_EVT_OK.

AT_EVT_RING Macro

File

at_parser.h

C

```
#define AT_EVT_RING 0
```

Description

This is macro AT_EVT_RING.

ATCMD_BUFFER_LEN Macro

File

at_parser.h

C

```
#define ATCMD_BUFFER_LEN 20
```

Description

This is macro ATCMD_BUFFER_LEN.

bt_max Macro

File

bufferutils.h

C

```
#define bt_max(a,b) (((a) > (b)) ? (a) : (b))
```

Description

This is macro bt_max.

bt_min Macro

File

bufferutils.h

C

```
#define bt_min(a,b) (((a) < (b)) ? (a) : (b))
```

Description

This is macro bt_min.

BUFFER_HDR_LEN Macro

File

buffer.h

C

```
#define BUFFER_HDR_LEN offsetof(bt_buffer, data)
```

Description

This is macro BUFFER_HDR_LEN.

BUFFER_STATE_FREE Macro

File

buffer.h

C

```
#define BUFFER_STATE_FREE 0
```

Description

for offsetof

BUFFER_STATE_USED Macro

File

buffer.h

C

```
#define BUFFER_STATE_USED 1
```

Description

This is macro BUFFER_STATE_USED.

max Macro

File

bufferutils.h

C

```
#define max(a,b) (using_max_is_dangerous = 0, use_bt_max_instead)
```

Description

This is macro max.

min Macro

File

bufferutils.h

C

```
#define min(a,b) (using_min_is_dangerous = 0, use_bt_min_instead)
```

Description

This is macro min.

VCARD_CALL_TYPE_DIALED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_DIALED 0
```

Description

This is macro VCARD_CALL_TYPE_DIALED.

VCARD_CALL_TYPE_MISSED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_MISSED 2
```

Description

This is macro VCARD_CALL_TYPE_MISSED.

VCARD_CALL_TYPE_RECEIVED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_RECEIVED 1
```

Description

This is macro VCARD_CALL_TYPE_RECEIVED.

VCARD_EVT_PROPERTY_PARAM Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_PARAM 3
```

Description

This is macro VCARD_EVT_PROPERTY_PARAM.

VCARD_EVT_PROPERTY_STARTED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_STARTED 2
```

Description

This is macro VCARD_EVT_PROPERTY_STARTED.

VCARD_EVT_PROPERTY_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_VALUE 4
```

Description

This is macro VCARD_EVT_PROPERTY_VALUE.

VCARD_EVT_VCARD_ENDED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_VCARD_ENDED 1
```

Description

This is macro VCARD_EVT_VCARD_ENDED.

VCARD_EVT_VCARD_STARTED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_VCARD_STARTED 0
```

Description

This is macro VCARD_EVT_VCARD_STARTED.

VCARD_PARAM_CALL_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_CALL_TYPE 0x80
```

Description

This is macro VCARD_PARAM_CALL_TYPE.

VCARD_PARAM_CELL Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_CELL 7
```

Description

This is macro VCARD_PARAM_CELL.

VCARD_PARAM_DIALED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_DIALED 4
```

Description

This is macro VCARD_PARAM_DIALED.

VCARD_PARAM_ENCODING Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_ENCODING 1
```

Description

This is macro VCARD_PARAM_ENCODING.

VCARD_PARAM_HOME Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_HOME 8
```

Description

This is macro VCARD_PARAM_HOME.

VCARD_PARAM_LANGUAGE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_LANGUAGE 3
```

Description

This is macro VCARD_PARAM_LANGUAGE.

VCARD_PARAM_MISSED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_MISSED 6
```

Description

This is macro VCARD_PARAM_MISSED.

VCARD_PARAM_RECEIVED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_RECEIVED 5
```

Description

This is macro VCARD_PARAM_RECEIVED.

VCARD_PARAM_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_TYPE 0
```

Description

This is macro VCARD_PARAM_TYPE.

VCARD_PARAM_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_VALUE 2
```

Description

This is macro VCARD_PARAM_VALUE.

VCARD_PARAM_WORK Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_WORK 9
```

Description

This is macro VCARD_PARAM_WORK.

VCARD_PARSER_STATE_READ_PARAM_NAME Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_PARAM_NAME 1
```

Description

This is macro VCARD_PARSER_STATE_READ_PARAM_NAME.

VCARD_PARSER_STATE_READ_PARAM_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_PARAM_VALUE 3
```

Description

This is macro VCARD_PARSER_STATE_READ_PARAM_VALUE.

VCARD_PARSER_STATE_READ_TYPE_NAME Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_TYPE_NAME 0
```

Description

This is macro VCARD_PARSER_STATE_READ_TYPE_NAME.

VCARD_PARSER_STATE_READ_TYPE_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_TYPE_VALUE 2
```

Description

This is macro VCARD_PARSER_STATE_READ_TYPE_VALUE.

VCARD_PARSER_STATE_SKIP_PARAM Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_SKIP_PARAM 4
```

Description

This is macro VCARD_PARSER_STATE_SKIP_PARAM.

VCARD_PARSER_STATE_SKIP_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_SKIP_TYPE 5
```

Description

This is macro VCARD_PARSER_STATE_SKIP_TYPE.

VCARD_PROPERTY_ADR Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ADR 7 // Delivery Address
```

Description

Delivery Address

VCARD_PROPERTY_AGENT Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_AGENT 17 // vCard of Person Representing
```

Description

vCard of Person Representing

VCARD_PROPERTY_BDAY Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_BDAY 6 // Birthday
```

Description

Birthday

VCARD_PROPERTY_BEGIN Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_BEGIN 0
```

Description

This is macro VCARD_PROPERTY_BEGIN.

VCARD_PROPERTY_CATEGORIES Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_CATEGORIES 26 // Categories
```

Description

Categories

VCARD_PROPERTY_CLASS Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_CLASS 28 // Class information
```

Description

Class information

VCARD_PROPERTY_EMAIL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_EMAIL 10 // Electronic Mail Address
```

Description

Electronic Mail Address

VCARD_PROPERTY_END Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_END 1
```

Description

This is macro VCARD_PROPERTY_END.

VCARD_PROPERTY_FN Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_FN 3 // Formatted Name
```

Description

Formatted Name

VCARD_PROPERTY_GEO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_GEO 13 // Geographic Position
```

Description

Geographic Position

VCARD_PROPERTY_KEY Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_KEY 24 // Public Encryption Key
```

Description

Public Encryption Key

VCARD_PROPERTY_LABEL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_LABEL 8 // Delivery
```

Description

Delivery

VCARD_PROPERTY_LOGO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_LOGO 16 // Organization Logo
```

Description

Organization Logo

VCARD_PROPERTY_MAILER Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_MAILER 11 // Electronic Mail
```

Description

Electronic Mail

VCARD_PROPERTY_N Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_N 4 // Structured Presentation of Name
```

Description

Structured Presentation of Name

VCARD_PROPERTY_NICKNAME Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_NICKNAME 25 // Nickname
```

Description

Nickname

VCARD_PROPERTY_NOTE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_NOTE 19 // Comments
```

Description

Comments

VCARD_PROPERTY_ORG Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ORG 18 // Name of Organization
```

Description

Name of Organization

VCARD_PROPERTY_PHOTO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_PHOTO 5 // Associated Image or Photo
```

Description

Associated Image or Photo

VCARD_PROPERTY_PROID Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_PROID 27 // Product ID
```

Description

Product ID

VCARD_PROPERTY_REV Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_REV 20 // Revision
```

Description

Revision

VCARD_PROPERTY_ROLE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ROLE 15 // Role within the Organization
```

Description

Role within the Organization

VCARD_PROPERTY_SORT_STRING Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_SORT_STRING 29 // String used for sorting operations
```

Description

String used for sorting operations

VCARD_PROPERTY_SOUND Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_SOUND 21 // Pronunciation of Name
```

Description

Pronunciation of Name

VCARD_PROPERTY_TEL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TEL 9 // Telephone Number
```

Description

Telephone Number

VCARD_PROPERTY_TITLE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TITLE 14 // Job
```

Description

Job

VCARD_PROPERTY_TZ Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TZ 12 // Time Zone
```

Description

Time Zone

VCARD_PROPERTY_UID Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_UID 23 // Unique ID
```

Description

Unique ID

VCARD_PROPERTY_URL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_URL 22 // Uniform Resource Locator
```

Description

Uniform Resource Locator

VCARD_PROPERTY_VERSION Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_VERSION 2 // vCard Version
```

Description

vCard Version

VCARD_PROPERTY_X_IRMC_CALL_DATETIME Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_X_IRMC_CALL_DATETIME 30 // Time stamp
```

Description

Time stamp

XML_EVT_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_EVT_ATTR_NAME 2
```

Description

This is macro XML_EVT_ATTR_NAME.

XML_EVT_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_EVT_ATTR_VALUE 3
```

Description

This is macro XML_EVT_ATTR_VALUE.

XML_EVT_TAG_ENDED Macro

File

xml_parser.h

C

```
#define XML_EVT_TAG_ENDED 1
```

Description

This is macro XML_EVT_TAG_ENDED.

XML_EVT_TAG_STARTED Macro

File

xml_parser.h

C

```
#define XML_EVT_TAG_STARTED 0
```

Description

This is macro XML_EVT_TAG_STARTED.

XML_PARSER_STATE_READ_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_ATTR_NAME 5
```

Description

This is macro XML_PARSER_STATE_READ_ATTR_NAME.

XML_PARSER_STATE_READ_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_ATTR_VALUE 8
```

Description

This is macro XML_PARSER_STATE_READ_ATTR_VALUE.

XML_PARSER_STATE_READ_TAG_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_TAG_NAME 1
```

Description

This is macro XML_PARSER_STATE_READ_TAG_NAME.

XML_PARSER_STATE_READ_TAG_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_TAG_VALUE 7
```

Description

This is macro XML_PARSER_STATE_READ_TAG_VALUE.

XML_PARSER_STATE_SKIP_ATTR Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_ATTR 9
```

Description

This is macro XML_PARSER_STATE_SKIP_ATTR.

XML_PARSER_STATE_SKIP_COMMENTS Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_COMMENTS 11
```

Description

This is macro XML_PARSER_STATE_SKIP_COMMENTS.

XML_PARSER_STATE_SKIP_PROC_INST Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_PROC_INST 10
```

Description

This is macro XML_PARSER_STATE_SKIP_PROC_INST.

XML_PARSER_STATE_SKIP_TAG Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_TAG 3
```

Description

This is macro XML_PARSER_STATE_SKIP_TAG.

XML_PARSER_STATE_WAIT_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_ATTR_NAME 4
```

Description

This is macro XML_PARSER_STATE_WAIT_ATTR_NAME.

XML_PARSER_STATE_WAIT_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_ATTR_VALUE 6
```

Description

This is macro XML_PARSER_STATE_WAIT_ATTR_VALUE.

XML_PARSER_STATE_WAIT_TAG_CLOSE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_TAG_CLOSE 2
```

Description

This is macro XML_PARSER_STATE_WAIT_TAG_CLOSE.

XML_PARSER_STATE_WAIT_TAG_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_TAG_NAME 0
```

Description

This is macro XML_PARSER_STATE_WAIT_TAG_NAME.

_readbn Macro

File

bufferutils.h

C

```
#define _readbn _readb
```

Description

This is macro _readbn.

_readui Macro

File

bufferutils.h

C

```
#define _readui(pvalue, pb, buf_len, offset) _readi((bt_int_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readui.

_readuin Macro

File

bufferutils.h

C

```
#define _readuin(pvalue, pb, buf_len, offset) _readin((bt_int_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readuin.

_readul Macro

File

bufferutils.h

C

```
#define _readul(pvalue, pb, buf_len, offset) _readl((bt_long_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readul.

_readuln Macro

File

bufferutils.h

C

```
#define _readuln(pvalue, pb, buf_len, offset) _readln((bt_long_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readuln.

_writebn Macro

File

bufferutils.h

C

```
#define _writebn _writeb
```

Description

This is macro _writebn.

bt_at_parser_callback_pf Type

File

at_parser.h

C

```
typedef void (* bt_at_parser_callback_pf)(struct _bt_at_parser_t* parser, bt_byte evt, void* evt_param,
void* callback_param);
```

Description

This is type bt_at_parser_callback_pf.

bt_at_parser_t Structure

File

at_parser.h

C

```
typedef struct _bt_at_parser_t {
  bt_byte state;
  bt_byte buffer[ATCMD_BUFFER_LEN + 1];
  bt_byte write_pos;
  bt_byte param_read_state;
  bt_byte param_read_nesting_level;
  bt_at_parser_callback_pf callback;
  void* callback_param;
} bt_at_parser_t;
```

Description

This is type bt_at_parser_t.

bt_bt_buffer_header_p Structure

File

buffer.h

C

```
typedef struct _bt_buffer_header_t {
  bt_int state;
} bt_buffer_header_t, * bt_bt_buffer_header_p;
```

Description

This is type bt_bt_buffer_header_p.

bt_buffer_header_t Structure

File

buffer.h

C

```
typedef struct _bt_buffer_header_t {
  bt_int state;
} bt_buffer_header_t, * bt_bt_buffer_header_p;
```

Description

This is type bt_buffer_header_t.

bt_buffer_mgr_p Structure

File

buffer.h

C

```
typedef struct _bt_buffer_mgr_t {
  bt_int max_buffers;
  bt_int buffer_size;
  bt_buffer_header_t* headers;
  bt_byte_p buffers;
} bt_buffer_mgr_t, * bt_buffer_mgr_p;
```

Description

This is type bt_buffer_mgr_p.

bt_buffer_mgr_t Structure

File

buffer.h

C

```
typedef struct _bt_buffer_mgr_t {
  bt_int max_buffers;
  bt_int buffer_size;
  bt_buffer_header_t* headers;
  bt_byte_p buffers;
} bt_buffer_mgr_t, * bt_buffer_mgr_p;
```

Description

This is type bt_buffer_mgr_t.

bt_packet_assembler_fp Type

File

packet.h

C

```
typedef bt_int (* bt_packet_assembler_fp)(struct _bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is type bt_packet_assembler_fp.

bt_packet_t Structure

File

packet.h

C

```
typedef struct _bt_packet_t {
  bt_packet_assembler_fp packet_assembler;
  const bt_byte* data;
  bt_int len;
  bt_byte state;
  bt_int data_pos;
  void* param;
} bt_packet_t;
```

Description

This is type bt_packet_t.

bt_queue_element_t Structure

File

queue.h

C

```
typedef struct _bt_queue_element_t {
  struct _bt_queue_element_t* next;
} bt_queue_element_t;
```

Description

This is type bt_queue_element_t.

bt_vcard_evt_prop_param_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_evt_prop_param_t {
  bt_byte param_id;
  bt_byte* param_value;
  bt_byte param_len;
} bt_vcard_evt_prop_param_t;
```

Description

This is type bt_vcard_evt_prop_param_t.

bt_vcard_evt_prop_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_evt_prop_t {
  bt_byte prop_id;
  bt_byte* prop_value;
  bt_byte value_len;
  bt_bool final;
} bt_vcard_evt_prop_t;
```

Description

This is type bt_vcard_evt_prop_t.

bt_vcard_parser_callback_fp Type

File

vcard_parser.h

C

```
typedef void (* bt_vcard_parser_callback_fp)(struct _bt_vcard_parser_t* parser, bt_byte evt, void*
evt_param, void* callback_param);
```

Description

This is type bt_vcard_parser_callback_fp.

bt_vcard_parser_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_parser_t {
  bt_byte state;
  bt_byte* buffer;
  bt_uint buffer_size;
  bt_byte write_pos;
  bt_byte prev_c;
  bt_byte cur_prop_id;
  bt_byte cur_param_id;
  bt_bool vcard_started;
  bt_vcard_parser_callback_fp callback;
  void* callback_param;
} bt_vcard_parser_t;
```

Description

This is type bt_vcard_parser_t.

bt_xml_evt_attribute_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_attribute_t {
  bt_byte* name;
  bt_byte len;
} bt_xml_evt_attribute_t;
```

Description

This is type bt_xml_evt_attribute_t.

bt_xml_evt_attribute_value_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_attribute_value_t {
  bt_byte* value;
  bt_byte len;
} bt_xml_evt_attribute_value_t;
```

Description

This is type bt_xml_evt_attribute_value_t.

bt_xml_evt_tag_started_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_tag_started_t {
  bt_byte* name;
  bt_byte len;
} bt_xml_evt_tag_started_t;
```

Description

This is type bt_xml_evt_tag_started_t.

bt_xml_parser_callback_pf Type

File

xml_parser.h

C

```
typedef void (* bt_xml_parser_callback_pf)(struct _bt_xml_parser_t* parser, bt_byte evt, void* evt_param,
void* callback_param);
```

Description

This is type bt_xml_parser_callback_pf.

bt_xml_parser_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_parser_t {
  bt_byte state;
  bt_byte* buffer;
  bt_uint buffer_size;
  bt_byte write_pos;
  bt_byte prev_c;
  bt_byte quote_c;
  bt_xml_parser_callback_pf callback;
  void* callback_param;
} bt_xml_parser_t;
```

Description

This is type bt_xml_parser_t.

Variables

Variables

	Name	Description
	_avctp_channels	This is variable _avctp_channels.
	_avctp_max_channels	This is variable _avctp_max_channels.
	_avctp_max_message_buffers	This is variable _avctp_max_message_buffers.
	_avctp_max_rx_message_len	This is variable _avctp_max_rx_message_len.
	_avctp_max_transports	This is variable _avctp_max_transports.
	_avctp_message_buffer_headers	This is variable _avctp_message_buffer_headers.
	_avctp_message_buffers	This is variable _avctp_message_buffers.
	_avctp_rx_buffers	This is variable _avctp_rx_buffers.
	_avctp_transports	This is variable _avctp_transports.
	_avdtp_cmd_buffer_headers	This is variable _avdtp_cmd_buffer_headers.
	_avdtp_cmd_buffers	This is variable _avdtp_cmd_buffers.
	_avdtp_cmd_param_buffers	This is variable _avdtp_cmd_param_buffers.
	_avdtp_codec_cfg_buffers	This is variable _avdtp_codec_cfg_buffers.
	_avdtp_control_channels	This is variable _avdtp_control_channels.
	_avdtp_l2cap_media_packet_buffer	This is variable _avdtp_l2cap_media_packet_buffer.
	_avdtp_listen_sep_buffers	This is variable _avdtp_listen_sep_buffers.
	_avdtp_max_cmd_buffers	This is variable _avdtp_max_cmd_buffers.
	_avdtp_max_cmd_param_len	This is variable _avdtp_max_cmd_param_len.
	_avdtp_max_codec_config_buffer_len	This is variable _avdtp_max_codec_config_buffer_len.
	_avdtp_max_control_channels	This is variable _avdtp_max_control_channels.
	_avdtp_max_seps	This is variable _avdtp_max_seps.
	_avdtp_max_streams	This is variable _avdtp_max_streams.

	_avdtp_max_transport_channels	This is variable _avdtp_max_transport_channels.
	_avdtp_max_tx_buffer_len	This is variable _avdtp_max_tx_buffer_len.
	_avdtp_rcv_sep_caps	This is variable _avdtp_rcv_sep_caps.
	_avdtp_rcv_sep_codec_cfg_buffer	This is variable _avdtp_rcv_sep_codec_cfg_buffer.
	_avdtp_sep_cfg_buffer_headers	This is variable _avdtp_sep_cfg_buffer_headers.
	_avdtp_sep_cfg_buffers	This is variable _avdtp_sep_cfg_buffers.
	_avdtp_seps	This is variable _avdtp_seps.
	_avdtp_streams	This is variable _avdtp_streams.
	_avdtp_transport_channels	This is variable _avdtp_transport_channels.
	_avdtp_tx_buffers	This is variable _avdtp_tx_buffers.
	_avrcp_channels	This is variable _avrcp_channels.
	_avrcp_cmd_buffer_headers	This is variable _avrcp_cmd_buffer_headers.
	_avrcp_cmd_buffers	This is variable _avrcp_cmd_buffers.
	_avrcp_cmd_param_buffers	This is variable _avrcp_cmd_param_buffers.
	_avrcp_cmd_timeout	This is variable _avrcp_cmd_timeout.
	_avrcp_device_name_buffers	This is variable _avrcp_device_name_buffers.
	_avrcp_devices_buffer	This is variable _avrcp_devices_buffer.
	_avrcp_max_channels	This is variable _avrcp_max_channels.
	_avrcp_max_cmd_buffers	This is variable _avrcp_max_cmd_buffers.
	_avrcp_max_cmd_param_len	This is variable _avrcp_max_cmd_param_len.
	_avrcp_max_device_name_len	This is variable _avrcp_max_device_name_len.
	_avrcp_max_search_results	This is variable _avrcp_max_search_results.
	_bt_avrcp_command_handler	This is variable _bt_avrcp_command_handler.
	_bt_avrcp_command_sent_handler	This is variable _bt_avrcp_command_sent_handler.
	_bt_avrcp_response_handler	This is variable _bt_avrcp_response_handler.
	_bt_avrcp_response_sent_handler	This is variable _bt_avrcp_response_sent_handler.
	_bt_log_level_max	This is variable _bt_log_level_max.
	_bt_log_level_min	This is variable _bt_log_level_min.
	_hci_event_handlers	This is variable _hci_event_handlers.
	_l2cap_cmd_assemblers	This is variable _l2cap_cmd_assemblers.
	_l2cap_cmd_parsers	This is variable _l2cap_cmd_parsers.
	_l2cap_request_handlers	This is variable _l2cap_request_handlers.
	_l2cap_response_handlers	This is variable _l2cap_response_handlers.
	_ram_size_avctp_buffers	This is variable _ram_size_avctp_buffers.
	_ram_size_avdtp_buffers	This is variable _ram_size_avdtp_buffers.
	_ram_size_avrcp_buffers	This is variable _ram_size_avrcp_buffers.
	_bt_hci_le_init	This is variable _bt_hci_le_init.
	_bt_ssp_evt_handler	This is variable _bt_ssp_evt_handler.
	_bt_ssp_init	This is variable _bt_ssp_init.
	_conn_state_recv_buffers	This is variable _conn_state_recv_buffers.
	_enable_local_config	if set to FALSE, l2cap_send_config will do nothing
	_enable_remote_config	This is variable _enable_remote_config.
	_frame_buffer_headers	This is variable _frame_buffer_headers.
	_frame_buffers	This is variable _frame_buffers.
	_hci_cmd_buffer_headers	Global variables defined by OEM configuration
	_hci_cmd_buffers	This is variable _hci_cmd_buffers.
	_hci_cmd_mgr	In hci_cmd_buffer.c
	_hci_cmd_param_buffers	This is variable _hci_cmd_param_buffers.
	_hci_connections	This is variable _hci_connections.
	_hci_enable_ctrl_to_host_flow_control	This is variable _hci_enable_ctrl_to_host_flow_control.
	_hci_enable_sco	This is variable _hci_enable_sco.
	_hci_evt_synch_connection_complete_handler	This is variable _hci_evt_synch_connection_complete_handler.
	_hci_l2cap_buffer_len	This is variable _hci_l2cap_buffer_len.
	_hci_linkkey_mgr	In hci_linkkey_buffer.c
	_hci_max_cmd_buffers	This is variable _hci_max_cmd_buffers.

	_hci_max_cmd_param_len	This is variable _hci_max_cmd_param_len.
	_hci_max_connect_attempts	This is variable _hci_max_connect_attempts.
	_hci_max_data_buffers	This is variable _hci_max_data_buffers.
	_hci_max_hci_connections	This is variable _hci_max_hci_connections.
	_hci_rcv_buffer_len	This is variable _hci_rcv_buffer_len.
	_hci_recv_sco_data_packet_fp	This is variable _hci_recv_sco_data_packet_fp.
	_hci_send_data_buffer_headers	This is variable _hci_send_data_buffer_headers.
	_hci_send_data_buffers	This is variable _hci_send_data_buffers.
	_hci_send_data_mgr	In hci_data_buffer.c
	_hci_tx_buffer_len	This is variable _hci_tx_buffer_len.
	_l2cap_channels	This is variable _l2cap_channels.
	_l2cap_cmd_buffer_headers	Global variables defined by OEM configuration
	_l2cap_cmd_buffers	This is variable _l2cap_cmd_buffers.
	_l2cap_cmd_frame_buffer	This is variable _l2cap_cmd_frame_buffer.
	_l2cap_cmd_frame_buffer_size	This is variable _l2cap_cmd_frame_buffer_size.
	_l2cap_connect_params	This is variable _l2cap_connect_params.
	_l2cap_connect_params_headers	This is variable _l2cap_connect_params_headers.
	_l2cap_eretr_handle_xmit_event_fp	This is variable _l2cap_eretr_handle_xmit_event_fp.
	_l2cap_eretr_pack_config_request_fp	This is variable _l2cap_eretr_pack_config_request_fp.
	_l2cap_eretr_recv_fp	This is variable _l2cap_eretr_recv_fp.
	_l2cap_eretr_send_data_fp	This is variable _l2cap_eretr_send_data_fp.
	_l2cap_eretr_send_smart_data_fp	This is variable _l2cap_eretr_send_smart_data_fp.
	_l2cap_fixed_channels	This is variable _l2cap_fixed_channels.
	_l2cap_hci_connect_packet_type	This is variable _l2cap_hci_connect_packet_type.
	_l2cap_hci_page_scan_repetition_mode	This is variable _l2cap_hci_page_scan_repetition_mode.
	_l2cap_hci_role_switch	This is variable _l2cap_hci_role_switch.
	_l2cap_idle_hci_connection_timeout	This is variable _l2cap_idle_hci_connection_timeout.
	_l2cap_max_channels	This is variable _l2cap_max_channels.
	_l2cap_max_cmd_buffers	This is variable _l2cap_max_cmd_buffers.
	_l2cap_max_fixed_channels	This is variable _l2cap_max_fixed_channels.
	_l2cap_max_psms	This is variable _l2cap_max_psms.
	_l2cap_psms	This is variable _l2cap_psms.
	_mgrs	In l2cap_mgr.c
	_pcmd_being_sent	In hci_send.c
	_pdata_being_sent	This is variable _pdata_being_sent.
	_phci_ctrl	In hci_param.c
	_ram_size_hci_buffers	This is variable _ram_size_hci_buffers.
	_ram_size_hci_cmd_queue	This is variable _ram_size_hci_cmd_queue.
	_ram_size_hci_linkkey_buffer	This is variable _ram_size_hci_linkkey_buffer.
	_ram_size_hci_param	This is variable _ram_size_hci_param.
	_ram_size_l2cap_buffers	This is variable _ram_size_l2cap_buffers.
	_ram_size_l2cap_mgr	This is variable _ram_size_l2cap_mgr.
	_ram_size_linkkey_storage	This is variable _ram_size_linkkey_storage.
	_ram_size_rfcomm_buffers	This is variable _ram_size_rfcomm_buffers.
	_ram_size_sdp_buffers	This is variable _ram_size_sdp_buffers.
	_ram_size_spp_buffers	This is variable _ram_size_spp_buffers.
	_recv_buffer	This is variable _recv_buffer.
	_resp_cq_head	This is variable _resp_cq_head.
	_rfcomm_channels	This is variable _rfcomm_channels.
	_rfcomm_cmd_buffer_headers	This is variable _rfcomm_cmd_buffer_headers.
	_rfcomm_cmd_buffers	This is variable _rfcomm_cmd_buffers.
	_rfcomm_data_buffer_headers	This is variable _rfcomm_data_buffer_headers.
	_rfcomm_data_buffers	This is variable _rfcomm_data_buffers.
	_rfcomm_dlcs	This is variable _rfcomm_dlcs.
	_rfcomm_enable_multidevice_channels	This is variable _rfcomm_enable_multidevice_channels.

	_rfcomm_info_len	This is variable _rfcomm_info_len.
	_rfcomm_local_credit	This is variable _rfcomm_local_credit.
	_rfcomm_local_credit_send_threshold	This is variable _rfcomm_local_credit_send_threshold.
	_rfcomm_max_channels	This is variable _rfcomm_max_channels.
	_rfcomm_max_cmd_buffers	This is variable _rfcomm_max_cmd_buffers.
	_rfcomm_max_data_buffers	This is variable _rfcomm_max_data_buffers.
	_rfcomm_max_dlcs	This is variable _rfcomm_max_dlcs.
	_rfcomm_max_sessions	This is variable _rfcomm_max_sessions.
	_rfcomm_pdu_size	This is variable _rfcomm_pdu_size.
	_rfcomm_sessions	Global variables defined by OEM configuration
	_sdp_client_max_buffers	This is variable _sdp_client_max_buffers.
	_sdp_client_packet_buffer_headers	Global variables defined by OEM configuration
	_sdp_client_packet_buffers	This is variable _sdp_client_packet_buffers.
	_sdp_found_attr_list_buffers	This is variable _sdp_found_attr_list_buffers.
	_sdp_found_attr_lists_buffers	This is variable _sdp_found_attr_lists_buffers.
	_sdp_found_sr_lists_buffers	This is variable _sdp_found_sr_lists_buffers.
	_sdp_max_attribute_result_len	This is variable _sdp_max_attribute_result_len.
	_sdp_max_buffers	This is variable _sdp_max_buffers.
	_sdp_max_search_result_len	This is variable _sdp_max_search_result_len.
	_sdp_packet_buffer_headers	This is variable _sdp_packet_buffer_headers.
	_sdp_packet_buffers	This is variable _sdp_packet_buffers.
	_sdp_service_tran_buffer_headers	This is variable _sdp_service_tran_buffer_headers.
	_sdp_service_tran_buffer_mgr	This is variable _sdp_service_tran_buffer_mgr.
	_sdp_service_tran_buffers	This is variable _sdp_service_tran_buffers.
	_sdp_start_fp	This is variable _sdp_start_fp.
	_sdp_tran_buffer_headers2	This is variable _sdp_tran_buffer_headers2.
	_sdp_tran_buffer_mgr2	This is variable _sdp_tran_buffer_mgr2.
	_sdp_tran_buffers2	This is variable _sdp_tran_buffers2.
	_send_buffer	This is variable _send_buffer.
	_send_cq_head	In hci_cmd_queue.c
	_spp_connect_device_address	This is variable _spp_connect_device_address.
	_spp_disable_buffering	This is variable _spp_disable_buffering.
	_spp_frame_buffers	This is variable _spp_frame_buffers.
	_spp_frame_len	This is variable _spp_frame_len.
	_spp_max_ports	This is variable _spp_max_ports.
	_spp_ports	This is variable _spp_ports.
	_spp_remaining_connect_attemtps	This is variable _spp_remaining_connect_attemtps.
	g_btx_ti_hci_callback	Data in btx_ti_shared_data.c
	g_btx_ti_hci_callback_param	This is variable g_btx_ti_hci_callback_param.
	sdp_db_main2	In sdp_server.c
	sdp_db_main2_len	This is variable sdp_db_main2_len.

Description

_avctp_channels Variable

File

avctp_private.h

C

```
bt_avctp_channel_t _avctp_channels[];
```

Description

This is variable _avctp_channels.

_avctp_max_channels Variable

File

avctp_private.h

C

```
const bt_byte _avctp_max_channels;
```

Description

This is variable _avctp_max_channels.

_avctp_max_message_buffers Variable

File

avctp_private.h

C

```
const bt_byte _avctp_max_message_buffers;
```

Description

This is variable _avctp_max_message_buffers.

_avctp_max_rx_message_len Variable

File

avctp_private.h

C

```
const bt_uint _avctp_max_rx_message_len;
```

Description

This is variable _avctp_max_rx_message_len.

_avctp_max_transports Variable

File

avctp_private.h

C

```
const bt_byte _avctp_max_transports;
```

Description

This is variable _avctp_max_transports.

_avctp_message_buffer_headers Variable

File

avctp_private.h

C

```
bt_buffer_header_t _avctp_message_buffer_headers[];
```

Description

This is variable _avctp_message_buffer_headers.

_avctp_message_buffers Variable

File

avctp_private.h

C

```
bt_avctp_message_t _avctp_message_buffers[];
```

Description

This is variable _avctp_message_buffers.

_avctp_rx_buffers Variable

File

avctp_private.h

C

```
bt_byte _avctp_rx_buffers[];
```

Description

This is variable _avctp_rx_buffers.

_avctp_transports Variable

File

avctp_private.h

C

```
bt_avctp_transport_t _avctp_transports[];
```

Description

This is variable _avctp_transports.

_avdtp_cmd_buffer_headers Variable

File

avdtp_private.h

C

```
bt_buffer_header_t _avdtp_cmd_buffer_headers[];
```

Description

This is variable _avdtp_cmd_buffer_headers.

_avdtp_cmd_buffers Variable

File

avdtp_private.h

C

```
bt_avdtp_control_cmd_t _avdtp_cmd_buffers[];
```

Description

This is variable _avdtp_cmd_buffers.

_avdtp_cmd_param_buffers Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_cmd_param_buffers[];
```

Description

This is variable _avdtp_cmd_param_buffers.

_avdtp_codec_cfg_buffers Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_codec_cfg_buffers[];
```

Description

This is variable _avdtp_codec_cfg_buffers.

_avdtp_control_channels Variable

File

avdtp_private.h

C

```
bt_avdtp_control_channel_t _avdtp_control_channels[];
```

Description

This is variable _avdtp_control_channels.

_avdtp_l2cap_media_packet_buffer Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_l2cap_media_packet_buffer[];
```

Description

This is variable _avdtp_l2cap_media_packet_buffer.

_avdtp_listen_sep_buffers Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_listen_sep_buffers[];
```

Description

This is variable _avdtp_listen_sep_buffers.

_avdtp_max_cmd_buffers Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_cmd_buffers;
```

Description

This is variable _avdtp_max_cmd_buffers.

_avdtp_max_cmd_param_len Variable

File

avdtp_private.h

C

```
const bt_uint _avdtp_max_cmd_param_len;
```

Description

This is variable _avdtp_max_cmd_param_len.

_avdtp_max_codec_config_buffer_len Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_codec_config_buffer_len;
```

Description

This is variable _avdtp_max_codec_config_buffer_len.

_avdtp_max_control_channels Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_control_channels;
```

Description

This is variable _avdtp_max_control_channels.

_avdtp_max_seps Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_seps;
```

Description

This is variable _avdtp_max_seps.

_avdtp_max_streams Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_streams;
```

Description

This is variable _avdtp_max_streams.

_avdtp_max_transport_channels Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_transport_channels;
```

Description

This is variable _avdtp_max_transport_channels.

_avdtp_max_tx_buffer_len Variable

File

avdtp_private.h

C

```
const bt_byte _avdtp_max_tx_buffer_len;
```

Description

This is variable _avdtp_max_tx_buffer_len.

_avdtp_rcv_sep_caps Variable

File

avdtp_private.h

C

```
bt_avdtp_sep_capabilities_t _avdtp_rcv_sep_caps;
```

Description

This is variable _avdtp_rcv_sep_caps.

_avdtp_rcv_sep_codec_cfg_buffer Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_rcv_sep_codec_cfg_buffer[];
```

Description

This is variable _avdtp_rcv_sep_codec_cfg_buffer.

_avdtp_sep_cfg_buffer_headers Variable

File

avdtp_private.h

C

```
bt_buffer_header_t _avdtp_sep_cfg_buffer_headers[];
```

Description

This is variable _avdtp_sep_cfg_buffer_headers.

_avdtp_sep_cfg_buffers Variable

File

avdtp_private.h

C

```
bt_avdtp_sep_capabilities_t _avdtp_sep_cfg_buffers[];
```

Description

This is variable _avdtp_sep_cfg_buffers.

_avdtp_seps Variable

File

avdtp_private.h

C

```
bt_avdtp_sep_t _avdtp_seps[];
```

Description

This is variable _avdtp_seps.

_avdtp_streams Variable

File

avdtp_private.h

C

```
bt_avdtp_stream_t _avdtp_streams[];
```

Description

This is variable _avdtp_streams.

_avdtp_transport_channels Variable

File

avdtp_private.h

C

```
bt_avdtp_transport_channel_t _avdtp_transport_channels[];
```

Description

This is variable _avdtp_transport_channels.

_avdtp_tx_buffers Variable

File

avdtp_private.h

C

```
bt_byte _avdtp_tx_buffers[];
```

Description

This is variable _avdtp_tx_buffers.

_avrcp_channels Variable

File

avrcp_private.h

C

```
bt_avrcp_channel_t _avrcp_channels[];
```

Description

This is variable _avrcp_channels.

_avrcp_cmd_buffer_headers Variable

File

avrcp_private.h

C

```
bt_buffer_header_t _avrcp_cmd_buffer_headers[];
```

Description

This is variable _avrcp_cmd_buffer_headers.

_avrcp_cmd_buffers Variable

File

avrcp_private.h

C

```
bt_av_command_t _avrcp_cmd_buffers[];
```

Description

This is variable _avrcp_cmd_buffers.

_avrcp_cmd_param_buffers Variable

File

avrcp_private.h

C

```
bt_byte _avrcp_cmd_param_buffers[];
```

Description

This is variable _avrcp_cmd_param_buffers.

_avrcp_cmd_timeout Variable

File

avrcp_private.h

C

```
const bt_byte _avrcp_cmd_timeout;
```

Description

This is variable _avrcp_cmd_timeout.

_avrcp_device_name_buffers Variable

File

avrcp_private.h

C

```
bt_byte _avrcp_device_name_buffers[];
```

Description

This is variable _avrcp_device_name_buffers.

_avrcp_devices_buffer Variable

File

avrcp_private.h

C

```
bt_avrcp_device_t _avrcp_devices_buffer[];
```

Description

This is variable _avrcp_devices_buffer.

_avrcp_max_channels Variable

File

avrcp_private.h

C

```
const bt_byte _avrcp_max_channels;
```

Description

This is variable _avrcp_max_channels.

_avrcp_max_cmd_buffers Variable

File

avrcp_private.h

C

```
const bt_byte _avrcp_max_cmd_buffers;
```

Description

This is variable _avrcp_max_cmd_buffers.

_avrcp_max_cmd_param_len Variable

File

avrcp_private.h

C

```
const bt_int _avrcp_max_cmd_param_len;
```

Description

This is variable _avrcp_max_cmd_param_len.

_avrcp_max_device_name_len Variable

File

avrcp_private.h

C

```
const bt_byte _avrcp_max_device_name_len;
```

Description

This is variable _avrcp_max_device_name_len.

_avrcp_max_search_results Variable

File

avrcp_private.h

C

```
const bt_byte _avrcp_max_search_results;
```

Description

This is variable _avrcp_max_search_results.

_bt_avrcp_command_handler Variable

File

avrcp_private.h

C

```
void (* _bt_avrcp_command_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_received_t* evt_param);
```

Description

This is variable _bt_avrcp_command_handler.

_bt_avrcp_command_sent_handler Variable

File

avrcp_private.h

C

```
void (* _bt_avrcp_command_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_sent_t* evt_param);
```

Description

This is variable _bt_avrcp_command_sent_handler.

_bt_avrcp_response_handler Variable

File

avrcp_private.h

C

```
void (* _bt_avrcp_response_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_received_t* evt_param);
```

Description

This is variable _bt_avrcp_response_handler.

_bt_avrcp_response_sent_handler Variable

File

avrcp_private.h

C

```
void (* _bt_avrcp_response_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_sent_t* evt_param);
```

Description

This is variable _bt_avrcp_response_sent_handler.

_bt_log_level_max Variable

File

bt_oem_config.h

C

```
bt_byte _bt_log_level_max = BT_LOG_LEVEL_MAX;
```

Description

This is variable _bt_log_level_max.

_bt_log_level_min Variable

File

bt_oem_config.h

C

```
bt_byte _bt_log_level_min = BT_LOG_LEVEL_MIN;
```

Description

This is variable _bt_log_level_min.

_hci_event_handlers Variable

File

hci_private.h

C

```
const bt_hci_event_handler_fp _hci_event_handlers[];
```

Description

This is variable _hci_event_handlers.

_l2cap_cmd_assemblers Variable

File

l2cap_private.h

C

```
const bt_l2cap_cmd_assembler_fp _l2cap_cmd_assemblers[];
```

Description

This is variable _l2cap_cmd_assemblers.

_l2cap_cmd_parsers Variable

File

l2cap_private.h

C

```
const bt_l2cap_cmd_parser_fp _l2cap_cmd_parsers[];
```

Description

This is variable _l2cap_cmd_parsers.

_l2cap_request_handlers Variable

File

l2cap_private.h

C

```
const bt_l2cap_request_handler_fp _l2cap_request_handlers[];
```

Description

This is variable _l2cap_request_handlers.

_l2cap_response_handlers Variable

File

l2cap_private.h

C

```
const bt_l2cap_response_handler_fp _l2cap_response_handlers[];
```

Description

This is variable _l2cap_response_handlers.

_ram_size_avctp_buffers Variable

File

avctp_private.h

C

```
const bt_uint _ram_size_avctp_buffers;
```

Description

This is variable _ram_size_avctp_buffers.

_ram_size_avdtp_buffers Variable

File

avdtp_private.h

C

```
const bt_uint _ram_size_avdtp_buffers;
```

Description

This is variable _ram_size_avdtp_buffers.

_ram_size_avrcp_buffers Variable

File

avrcp_private.h

C

```
const bt_uint _ram_size_avrcp_buffers;
```

Description

This is variable _ram_size_avrcp_buffers.

_bt_hci_le_init Variable

File

hci_private.h

C

```
void (* _bt_hci_le_init)(bt_hci_le_ctrl_state_t* le_ctrl_state);
```

Description

This is variable _bt_hci_le_init.

_bt_ssp_evt_handler Variable

File

ssp.h

C

```
void (* _bt_ssp_evt_handler)(bt_hci_event_t* evt);
```

Description

This is variable _bt_ssp_evt_handler.

_bt_ssp_init Variable

File

ssp.h

C

```
void (* _bt_ssp_init)(void);
```

Description

This is variable _bt_ssp_init.

_conn_state_recv_buffers Variable

File

hci_private.h

C

```
bt_byte _conn_state_recv_buffers[];
```

Description

This is variable _conn_state_recv_buffers.

_enable_local_config Variable

File

l2cap_test.h

C

```
bt_bool _enable_local_config;
```

Description

if set to FALSE, l2cap_send_config will do nothing

_enable_remote_config Variable

File

l2cap_test.h

C

```
bt_bool _enable_remote_config;
```

Description

This is variable _enable_remote_config.

_frame_buffer_headers Variable

File

l2cap_private.h

C

```
bt_buffer_header_t _frame_buffer_headers[];
```

Description

This is variable _frame_buffer_headers.

_frame_buffers Variable

File

l2cap_private.h

C

```
bt_byte _frame_buffers[];
```

Description

This is variable _frame_buffers.

_hci_cmd_buffer_headers Variable

File

hci_private.h

C

```
bt_buffer_header_t _hci_cmd_buffer_headers[];
```

Description

Global variables defined by OEM configuration

_hci_cmd_buffers Variable

File

hci_private.h

C

```
bt_hci_command_t _hci_cmd_buffers[];
```

Description

This is variable _hci_cmd_buffers.

_hci_cmd_mgr Variable

File

hci_private.h

C

```
bt_buffer_mgr_t _hci_cmd_mgr;
```

Description

In hci_cmd_buffer.c

_hci_cmd_param_buffers Variable

File

hci_private.h

C

```
bt_byte _hci_cmd_param_buffers[];
```

Description

This is variable _hci_cmd_param_buffers.

_hci_connections Variable

File

hci_private.h

C

```
bt_hci_conn_state_t _hci_connections[];
```

Description

This is variable _hci_connections.

_hci_enable_ctrl_to_host_flow_control Variable

File

hci_private.h

C

```
const bt_bool _hci_enable_ctrl_to_host_flow_control;
```

Description

This is variable _hci_enable_ctrl_to_host_flow_control.

_hci_enable_sco Variable

File

hci_private.h

C

```
const bt_bool _hci_enable_sco;
```

Description

This is variable _hci_enable_sco.

_hci_evt_synch_connection_complete_handler Variable

File

hci_private.h

C

```
bt_hci_event_handler_fp _hci_evt_synch_connection_complete_handler;
```

Description

This is variable _hci_evt_synch_connection_complete_handler.

_hci_l2cap_buffer_len Variable

File

hci_private.h

C

```
const bt_uint _hci_l2cap_buffer_len;
```

Description

This is variable _hci_l2cap_buffer_len.

_hci_linkkey_mgr Variable

File

hci_private.h

C

```
bt_buffer_mgr_t _hci_linkkey_mgr;
```

Description

In hci_linkkey_buffer.c

_hci_max_cmd_buffers Variable

File

hci_private.h

C

```
const bt_byte _hci_max_cmd_buffers;
```

Description

This is variable _hci_max_cmd_buffers.

_hci_max_cmd_param_len Variable

File

hci_private.h

C

```
const bt_byte _hci_max_cmd_param_len;
```

Description

This is variable _hci_max_cmd_param_len.

_hci_max_connect_attempts Variable

File

hci_private.h

C

```
const bt_byte _hci_max_connect_attempts;
```

Description

This is variable _hci_max_connect_attempts.

_hci_max_data_buffers Variable

File

hci_private.h

C

```
const bt_byte _hci_max_data_buffers;
```

Description

This is variable _hci_max_data_buffers.

_hci_max_hci_connections Variable

File

hci_private.h

C

```
const bt_byte _hci_max_hci_connections;
```

Description

This is variable _hci_max_hci_connections.

_hci_rcv_buffer_len Variable

File

hci_private.h

C

```
const bt_uint _hci_rcv_buffer_len;
```

Description

This is variable _hci_rcv_buffer_len.

_hci_recv_sco_data_packet_fp Variable

File

hci_private.h

C

```
void (* _hci_recv_sco_data_packet_fp)(bt_byte* pbuf);
```

Description

This is variable _hci_recv_sco_data_packet_fp.

_hci_send_data_buffer_headers Variable

File

hci_private.h

C

```
bt_buffer_header_t _hci_send_data_buffer_headers[];
```

Description

This is variable _hci_send_data_buffer_headers.

_hci_send_data_buffers Variable

File

hci_private.h

C

```
bt_hci_data_t _hci_send_data_buffers[];
```

Description

This is variable _hci_send_data_buffers.

_hci_send_data_mgr Variable

File

hci_private.h

C

```
bt_buffer_mgr_t _hci_send_data_mgr;
```

Description

In hci_data_buffer.c

_hci_tx_buffer_len Variable

File

hci_private.h

C

```
const bt_uint _hci_tx_buffer_len;
```

Description

This is variable _hci_tx_buffer_len.

_l2cap_channels Variable

File

l2cap_private.h

C

```
bt_l2cap_channel_t _l2cap_channels[];
```

Description

This is variable _l2cap_channels.

_l2cap_cmd_buffer_headers Variable

File

l2cap_private.h

C

```
bt_buffer_header_t _l2cap_cmd_buffer_headers[];
```

Description

Global variables defined by OEM configuration

_l2cap_cmd_buffers Variable

File

l2cap_private.h

C

```
bt_l2cap_command_t _l2cap_cmd_buffers[];
```

Description

This is variable _l2cap_cmd_buffers.

_l2cap_cmd_frame_buffer Variable

File

l2cap_private.h

C

```
bt_byte _l2cap_cmd_frame_buffer[];
```

Description

This is variable _l2cap_cmd_frame_buffer.

_l2cap_cmd_frame_buffer_size Variable

File

l2cap_private.h

C

```
bt_uint _l2cap_cmd_frame_buffer_size;
```

Description

This is variable _l2cap_cmd_frame_buffer_size.

_l2cap_connect_params Variable

File

l2cap_private.h

C

```
bt_l2cap_connect_params_t _l2cap_connect_params[];
```

Description

This is variable _l2cap_connect_params.

_l2cap_connect_params_headers Variable

File

l2cap_private.h

C

```
bt_buffer_header_t _l2cap_connect_params_headers[];
```

Description

This is variable _l2cap_connect_params_headers.

_l2cap_eretr_handle_xmit_event_fp Variable

File

l2cap_private.h

C

```
bt_bool (* _l2cap_eretr_handle_xmit_event_fp)(bt_l2cap_xmit_event_param_t* param);
```

Description

This is variable _l2cap_eretr_handle_xmit_event_fp.

_l2cap_eretr_pack_config_request_fp Variable

File

l2cap_private.h

C

```
void (* _l2cap_eretr_pack_config_request_fp)(bt_l2cap_channel_t* channel, bt_byte* buffer, bt_int
buffer_len, bt_int* offset);
```

Description

This is variable _l2cap_eretr_pack_config_request_fp.

_l2cap_eretr_recv_fp Variable

File

l2cap_private.h

C

```
void (* _l2cap_eretr_recv_fp)(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte* pdata, bt_int len);
```

Description

This is variable _l2cap_eretr_recv_fp.

_l2cap_eretr_send_data_fp Variable

File

l2cap_private.h

C

```
bt_bool (* _l2cap_eretr_send_data_fp)(bt_l2cap_channel_t* pch, bt_byte* data, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is variable _l2cap_eretr_send_data_fp.

_l2cap_eretr_send_smart_data_fp Variable

File

l2cap_private.h

C

```
bt_bool (* _l2cap_eretr_send_smart_data_fp)(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is variable _l2cap_eretr_send_smart_data_fp.

_l2cap_fixed_channels Variable

File

l2cap_private.h

C

```
bt_l2cap_fixed_channel_t* _l2cap_fixed_channels;
```

Description

This is variable _l2cap_fixed_channels.

_l2cap_hci_connect_packet_type Variable

File

l2cap_private.h

C

```
bt_uint _l2cap_hci_connect_packet_type;
```

Description

This is variable _l2cap_hci_connect_packet_type.

_l2cap_hci_page_scan_repetition_mode Variable

File

l2cap_private.h

C

```
bt_byte _l2cap_hci_page_scan_repetition_mode;
```

Description

This is variable _l2cap_hci_page_scan_repetition_mode.

_l2cap_hci_role_switch Variable

File

l2cap_private.h

C

```
bt_byte _l2cap_hci_role_switch;
```

Description

This is variable _l2cap_hci_role_switch.

_l2cap_idle_hci_connection_timeout Variable

File

l2cap_private.h

C

```
bt_long _l2cap_idle_hci_connection_timeout;
```

Description

This is variable _l2cap_idle_hci_connection_timeout.

_l2cap_max_channels Variable

File

l2cap_private.h

C

```
const bt_byte _l2cap_max_channels;
```

Description

This is variable _l2cap_max_channels.

_l2cap_max_cmd_buffers Variable

File

l2cap_private.h

C

```
const bt_byte _l2cap_max_cmd_buffers;
```

Description

This is variable _l2cap_max_cmd_buffers.

_l2cap_max_fixed_channels Variable

File

l2cap_private.h

C

```
bt_byte _l2cap_max_fixed_channels;
```

Description

This is variable _l2cap_max_fixed_channels.

_l2cap_max_psms Variable

File

l2cap_private.h

C

```
const bt_byte _l2cap_max_psms;
```

Description

This is variable _l2cap_max_psms.

_l2cap_psms Variable

File

l2cap_private.h

C

```
bt_l2cap_psm_t _l2cap_psms[];
```

Description

This is variable _l2cap_psms.

_mgrs Variable

File

l2cap_private.h

C

```
bt_l2cap_mgr_t _mgrs[];
```

Description

In l2cap_mgr.c

_pcmd_being_sent Variable

File

hci_private.h

C

```
bt_hci_command_p _pcmd_being_sent;
```

Description

In hci_send.c

_pdata_being_sent Variable

File

hci_private.h

C

```
bt_hci_data_p _pdata_being_sent;
```

Description

This is variable _pdata_being_sent.

_phci_ctrl Variable

File

hci_private.h

C

```
bt_hci_ctrl_state_t* _phci_ctrl;
```

Description

In hci_param.c

_ram_size_hci_buffers Variable

File

hci_private.h

C

```
const bt_uint _ram_size_hci_buffers;
```

Description

This is variable _ram_size_hci_buffers.

_ram_size_hci_cmd_queue Variable

File

hci_private.h

C

```
const bt_int _ram_size_hci_cmd_queue;
```

Description

This is variable _ram_size_hci_cmd_queue.

_ram_size_hci_linkkey_buffer Variable

File

hci_private.h

C

```
const bt_int _ram_size_hci_linkkey_buffer;
```

Description

This is variable _ram_size_hci_linkkey_buffer.

_ram_size_hci_param Variable

File

hci_private.h

C

```
const bt_int _ram_size_hci_param;
```

Description

This is variable _ram_size_hci_param.

_ram_size_l2cap_buffers Variable

File

l2cap_private.h

C

```
const bt_uint _ram_size_l2cap_buffers;
```

Description

This is variable _ram_size_l2cap_buffers.

_ram_size_l2cap_mgr Variable

File

l2cap_private.h

C

```
const bt_int _ram_size_l2cap_mgr;
```

Description

This is variable _ram_size_l2cap_mgr.

_ram_size_linkkey_storage Variable

File

hci_private.h

C

```
const bt_int _ram_size_linkkey_storage;
```

Description

This is variable _ram_size_linkkey_storage.

_ram_size_rfcomm_buffers Variable

File

rfcomm_private.h

C

```
const bt_uint _ram_size_rfcomm_buffers;
```

Description

This is variable _ram_size_rfcomm_buffers.

_ram_size_sdp_buffers Variable

File

sdp_private.h

C

```
const bt_uint _ram_size_sdp_buffers;
```

Description

This is variable _ram_size_sdp_buffers.

_ram_size_spp_buffers Variable

File

spp_private.h

C

```
const bt_uint _ram_size_spp_buffers;
```

Description

This is variable _ram_size_spp_buffers.

_recv_buffer Variable

File

hci_private.h

C

```
bt_byte _recv_buffer[];
```

Description

This is variable _recv_buffer.

_resp_cq_head Variable

File

hci_private.h

C

```
bt_queue_element_t* _resp_cq_head;
```

Description

This is variable _resp_cq_head.

_rfcomm_channels Variable

File

rfcomm_private.h

C

```
bt_rfcomm_server_channel_t _rfcomm_channels[];
```

Description

This is variable _rfcomm_channels.

_rfcomm_cmd_buffer_headers Variable

File

rfcomm_private.h

C

```
bt_buffer_header_t _rfcomm_cmd_buffer_headers[];
```

Description

This is variable _rfcomm_cmd_buffer_headers.

_rfcomm_cmd_buffers Variable

File

rfcomm_private.h

C

```
bt_rfcomm_command_t _rfcomm_cmd_buffers[];
```

Description

This is variable _rfcomm_cmd_buffers.

_rfcomm_data_buffer_headers Variable

File

rfcomm_private.h

C

```
bt_buffer_header_t _rfcomm_data_buffer_headers[];
```

Description

This is variable _rfcomm_data_buffer_headers.

_rfcomm_data_buffers Variable

File

rfcomm_private.h

C

```
bt_byte _rfcomm_data_buffers[];
```

Description

This is variable _rfcomm_data_buffers.

_rfcomm_dlcs Variable

File

rfcomm_private.h

C

```
bt_rfcomm_dlc_t _rfcomm_dlcs[];
```

Description

This is variable _rfcomm_dlcs.

_rfcomm_enable_multidevice_channels Variable

File

rfcomm_private.h

C

```
const bt_bool _rfcomm_enable_multidevice_channels;
```

Description

This is variable _rfcomm_enable_multidevice_channels.

_rfcomm_info_len Variable

File

rfcomm_private.h

C

```
const bt_uint _rfcomm_info_len;
```

Description

This is variable _rfcomm_info_len.

_rfcomm_local_credit Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_local_credit;
```

Description

This is variable _rfcomm_local_credit.

_rfcomm_local_credit_send_threshold Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_local_credit_send_threshold;
```

Description

This is variable _rfcomm_local_credit_send_threshold.

_rfcomm_max_channels Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_max_channels;
```

Description

This is variable _rfcomm_max_channels.

_rfcomm_max_cmd_buffers Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_max_cmd_buffers;
```

Description

This is variable _rfcomm_max_cmd_buffers.

_rfcomm_max_data_buffers Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_max_data_buffers;
```

Description

This is variable _rfcomm_max_data_buffers.

_rfcomm_max_dlcs Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_max_dlcs;
```

Description

This is variable _rfcomm_max_dlcs.

_rfcomm_max_sessions Variable

File

rfcomm_private.h

C

```
const bt_byte _rfcomm_max_sessions;
```

Description

This is variable _rfcomm_max_sessions.

_rfcomm_pdu_size Variable

File

rfcomm_private.h

C

```
const bt_uint _rfcomm_pdu_size;
```

Description

This is variable _rfcomm_pdu_size.

_rfcomm_sessions Variable

File

rfcomm_private.h

C

```
bt_rfcomm_session_t _rfcomm_sessions[];
```

Description

Global variables defined by OEM configuration

_sdp_client_max_buffers Variable

File

sdp_private.h

C

```
const bt_byte _sdp_client_max_buffers;
```

Description

This is variable _sdp_client_max_buffers.

_sdp_client_packet_buffer_headers Variable

File

sdp_private.h

C

```
bt_buffer_header_t _sdp_client_packet_buffer_headers[];
```

Description

Global variables defined by OEM configuration

_sdp_client_packet_buffers Variable

File

sdp_private.h

C

```
bt_sdp_packet_t _sdp_client_packet_buffers[];
```

Description

This is variable _sdp_client_packet_buffers.

_sdp_found_attr_list_buffers Variable

File

sdp_private.h

C

```
bt_byte* _sdp_found_attr_list_buffers[];
```

Description

This is variable _sdp_found_attr_list_buffers.

_sdp_found_attr_lists_buffers Variable

File

sdp_private.h

C

```
bt_sdp_found_attr_list_t _sdp_found_attr_lists_buffers[];
```

Description

This is variable _sdp_found_attr_lists_buffers.

_sdp_found_sr_lists_buffers Variable

File

sdp_private.h

C

```
bt_sr_handle_t _sdp_found_sr_lists_buffers[];
```

Description

This is variable _sdp_found_sr_lists_buffers.

_sdp_max_attribute_result_len Variable

File

sdp_private.h

C

```
const bt_uint _sdp_max_attribute_result_len;
```

Description

This is variable _sdp_max_attribute_result_len.

_sdp_max_buffers Variable

File

sdp_private.h

C

```
const bt_byte _sdp_max_buffers;
```

Description

This is variable _sdp_max_buffers.

_sdp_max_search_result_len Variable

File

sdp_private.h

C

```
const bt_uint _sdp_max_search_result_len;
```

Description

This is variable _sdp_max_search_result_len.

_sdp_packet_buffer_headers Variable

File

sdp_private.h

C

```
bt_buffer_header_t _sdp_packet_buffer_headers[];
```

Description

This is variable _sdp_packet_buffer_headers.

_sdp_packet_buffers Variable

File

sdp_private.h

C

```
bt_sdp_packet_t _sdp_packet_buffers[];
```

Description

This is variable _sdp_packet_buffers.

_sdp_service_tran_buffer_headers Variable

File

sdp_private.h

C

```
bt_buffer_header_t _sdp_service_tran_buffer_headers[];
```

Description

This is variable _sdp_service_tran_buffer_headers.

_sdp_service_tran_buffer_mgr Variable

File

sdp_private.h

C

```
bt_buffer_mgr_t _sdp_service_tran_buffer_mgr;
```

Description

This is variable _sdp_service_tran_buffer_mgr.

_sdp_service_tran_buffers Variable

File

sdp_private.h

C

```
bt_sdp_service_transaction_t _sdp_service_tran_buffers[];
```

Description

This is variable _sdp_service_tran_buffers.

_sdp_start_fp Variable

File

sdp_private.h

C

```
bt_bool (* _sdp_start_fp)(bt_l2cap_mgr_p l2cap_mgr, const bt_byte* sdp_db, bt_uint sdp_db_len);
```

Description

This is variable _sdp_start_fp.

_sdp_tran_buffer_headers2 Variable

File

sdp_private.h

C

```
bt_buffer_header_t _sdp_tran_buffer_headers2[];
```

Description

This is variable _sdp_tran_buffer_headers2.

_sdp_tran_buffer_mgr2 Variable

File

sdp_private.h

C

```
bt_buffer_mgr_t _sdp_tran_buffer_mgr2;
```

Description

This is variable _sdp_tran_buffer_mgr2.

_sdp_tran_buffers2 Variable

File

sdp_private.h

C

```
bt_sdp_transaction_t _sdp_tran_buffers2[];
```

Description

This is variable _sdp_tran_buffers2.

_send_buffer Variable

File

hci_private.h

C

```
bt_byte _send_buffer[];
```

Description

This is variable _send_buffer.

_send_cq_head Variable

File

hci_private.h

C

```
bt_queue_element_t* _send_cq_head;
```

Description

In hci_cmd_queue.c

_spp_connect_device_address Variable

File

spp_private.h

C

```
bt_bdaddr_t _spp_connect_device_address;
```

Description

This is variable _spp_connect_device_address.

_spp_disable_buffering Variable

File

spp_private.h

C

```
const bt_byte _spp_disable_buffering;
```

Description

This is variable _spp_disable_buffering.

_spp_frame_buffers Variable

File

spp_private.h

C

```
bt_byte* _spp_frame_buffers;
```

Description

This is variable _spp_frame_buffers.

_spp_frame_len Variable

File

spp_private.h

C

```
bt_int* _spp_frame_len;
```

Description

This is variable _spp_frame_len.

_spp_max_ports Variable

File

spp_private.h

C

```
const bt_byte _spp_max_ports;
```

Description

This is variable _spp_max_ports.

_spp_ports Variable

File

spp_private.h

C

```
bt_spp_port_t _spp_ports[];
```

Description

This is variable _spp_ports.

_spp_remaining_connect_attemtps Variable

File

spp_private.h

C

```
bt_byte _spp_remaining_connect_attemtps;
```

Description

This is variable _spp_remaining_connect_attemtps.

g_btx_ti_hci_callback Variable

File

ti_private.h

C

```
bt_hci_cmd_callback_fp g_btx_ti_hci_callback;
```

Description

Data in btx_ti_shared_data.c

g_btx_ti_hci_callback_param Variable

File

ti_private.h

C

```
void* g_btx_ti_hci_callback_param;
```

Description

This is variable g_btx_ti_hci_callback_param.

sdp_db_main2 Variable

File

sdp_private.h

C

```
const bt_byte* sdp_db_main2;
```

Description

In sdp_server.c

sdp_db_main2_len Variable

File

sdp_private.h

C

```
bt_uint sdp_db_main2_len;
```

Description

This is variable sdp_db_main2_len.

Files

Files

Name	Description
a2dp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_aac.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_mpeg.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_sbc.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_control.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_command.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_config_event_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
avrcp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
at_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_bdaddr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
baseband.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_hcitr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_log.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_oem.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_oem_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_std.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_storage.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_system.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_types.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bufferutils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
channel.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
chmanager.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
cmdbuffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
command.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

csr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
frame_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
gap.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
hci.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_cmd_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_cmd_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_config_event_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_conn_state.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_ctrl_state.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_data_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_data_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_eir.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_errors.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_evt_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_le.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
hci_linkkey_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_transport.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_utils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_3wire.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_bcsp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_tih4.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_uart.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

lm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_command_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_config_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_eretr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_fixed_channel.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
l2cap_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
l2cap_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_psm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_test.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
patch.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_cmd_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_mgr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_mx.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sbc.h	Bluetooth low-complexity, subband codec (SBC) library Copyright (C) 2008-2010 Nokia Corporation Copyright (C) 2004-2010 Marcel Holtmann Copyright (C) 2004-2005 Henryk Ploetz Copyright (C) 2005-2006 Brad Midgley This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU... more
sdp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_client.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
sdp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_utils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ssp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ssp_event.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ssp_event_handler.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ti.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
ti_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
types.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
vcard_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
xml_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

Description

a2dp.h

Functions

	Name	Description
	bt_a2dp_find_server	This is function bt_a2dp_find_server.
	bt_a2dp_find_sink	brief Find sink ingroup a2dp details This function looks for a sink on a remote device specified by c deviceAddress and, if found, returns features supported by the sink. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more

	bt_a2dp_find_source	brief Find source ingroup a2dp details This function looks for a source on a remote device specified by c deviceAddress and, if found, returns features supported by the source. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_get_mgr	brief Return a pointer to an instance of the A2DP manager. ingroup a2dp details This function returns a pointer to an instance of the A@DP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.
	bt_a2dp_init	brief Initialize the A2DP layer. ingroup a2dp details This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.
	bt_a2dp_open_and_start_stream	brief Open & start a stream ingroup a2dp details Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with... more
	bt_a2dp_register_aac_codec	brief Register default AAC codec ingroup a2dp details This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_register_callback	brief Register a A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.... more
	bt_a2dp_register_mpeg_codec	brief Register default MPEG codec ingroup a2dp details This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_set_configuration	This is function bt_a2dp_set_configuration.
	bt_a2dp_start	brief Start the A2DP layer. ingroup a2dp details This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	__A2DP_H	This is macro __A2DP_H.
	A2DP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	A2DP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.

	A2DP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	A2DP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	A2DP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	A2DP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	A2DP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	A2DP_EVT_DISCOVER_SEP_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	A2DP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	A2DP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	A2DP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	A2DP_EVT_NOTHING	This is macro A2DP_EVT_NOTHING.
	A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED	< This event is generated when a local device completed "open and start" request.
	A2DP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	A2DP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	A2DP_EVT_RECONFIGURE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	A2DP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	A2DP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	A2DP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION	This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.
	A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	A2DP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	A2DP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	A2DP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	A2DP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	A2DP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.

	A2DP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.
	A2DP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	A2DP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	A2DP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	A2DP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	A2DP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	A2DP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	A2DP_MANAGER_STATE_CONNECTING	This is macro A2DP_MANAGER_STATE_CONNECTING.
	A2DP_MANAGER_STATE_IDLE	This is macro A2DP_MANAGER_STATE_IDLE.
	A2DP_SINK_FEATURE_AMPLIFIER	< Amplifier
	A2DP_SINK_FEATURE_HEADPHONE	< Headphone
	A2DP_SINK_FEATURE_RECORDER	< Recorder
	A2DP_SINK_FEATURE_SPEAKER	< Speaker
	A2DP_SOURCE_FEATURE_MICROPHONE	< Mic
	A2DP_SOURCE_FEATURE_MIXER	< Mixer
	A2DP_SOURCE_FEATURE_PLAYER	< Player
	A2DP_SOURCE_FEATURE_TUNER	< Tuner
	bt_a2dp_abort_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_a2dp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more

	bt_a2dp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up... more
	bt_a2dp_call_codec	This is macro bt_a2dp_call_codec.
	bt_a2dp_cancel_listen	brief Cancel listening for incoming connections. ingroup a2dp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_clear_media_tx_queue	brief Clear send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_a2dp_clear_media_tx_queue. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_a2dp_close_stream	brief Close a stream. ingroup a2dp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_a2dp_connect	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more

	bt_a2dp_connect_ex	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more
	bt_a2dp_create_stream	brief Create a stream. ingroup a2dp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr A2DP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_a2dp_destroy_stream	brief Destroy a stream. ingroup a2dp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_disconnect	brief Disconnect from a remote device. ingroup a2dp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_a2dp_discover	brief Discover SEPs on a remote device. ingroup a2dp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr A2DP... more
	bt_a2dp_find_codec	brief Find a codec ingroup a2dp details A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in... more

	bt_a2dp_get_all_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_configuration	This is macro bt_a2dp_get_configuration.
	bt_a2dp_get_hci_connection	brief Get HCI connection for a stream ingroup a2dp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param mgr A2DP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p... more
	bt_a2dp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup a2dp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr A2DP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more

	bt_a2dp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup a2dp details This function returns the type of the codec currently used with the stream. param mgr A2DP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_a2dp_get_stream_config	brief Get stream's configuration. ingroup a2dp details This function returns a pointer to a structure holding current configuration of the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_a2dp_get_stream_direction	This is macro bt_a2dp_get_stream_direction.
	bt_a2dp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup a2dp details This function returns the ID of the local SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_remote_address	brief Get stream's remote BT address. ingroup a2dp details This function returns the address of the remote device associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_a2dp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup a2dp details This function returns the ID of the remote SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_state	brief Get local stream state. ingroup a2dp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr A2DP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more

	bt_a2dp_listen	brief Listen for incoming connections. ingroup a2dp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more
	bt_a2dp_open_stream	This is macro bt_a2dp_open_stream.
	bt_a2dp_reconfigure_stream	brief Reconfigure stream. ingroup a2dp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_a2dp_register_sink	brief Register a Sink SEP with the local A2DP manager. ingroup a2dp details This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_register_source	brief Register a Source SEP with the local A2DP manager. ingroup a2dp details This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling ::bt_a2dp_remove_media_tx_buffer. param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more
	bt_a2dp_report_delay	This is macro bt_a2dp_report_delay.

	bt_a2dp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more
	bt_a2dp_start_stream	brief Start a stream. ingroup a2dp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more
	bt_a2dp_suspend_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more

Structures

	Name	Description
	_bt_a2dp_evt_open_and_start_stream_completed_s	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	_bt_a2dp_mgr_t	brief A2DP manager. ingroup a2dp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with ::bt_a2dp_get_mgr.
	bt_a2dp_evt_open_and_start_stream_completed_t	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.

Types

	Name	Description
	bt_a2dp_event_t	This is type bt_a2dp_event_t.

	bt_a2dp_find_server_callback_fp	brief Notify the application of the result of searching for a remote A2DP entity (sourse or sink) ingroup a2dp details This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed. param supported_features Features supported by a remote A2DP entity. param found c TRUE if an A2DP entity has been found on the remote device. c FALSE otherwise. param param pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its c callback_param parameter.
	bt_a2dp_mgr_callback_fp	brief A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr A2DP manager. param evt A2DP event. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local... more
	bt_a2dp_mgr_t	This is type bt_a2dp_mgr_t.

Unions

	Name	Description
	_bt_a2dp_event_u	brief Parameter to an application callback. ingroup a2dp details This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_aac.h

Functions

	Name	Description
	bt_a2dp_aac_codec_handler	This is function bt_a2dp_aac_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_AAC_H	This is macro __A2DP_CODEC_AAC_H.
	AAC_CHANNELS_1	This is macro AAC_CHANNELS_1.
	AAC_CHANNELS_2	This is macro AAC_CHANNELS_2.
	AAC_CHANNELS_ALL	This is macro AAC_CHANNELS_ALL.
	AAC_OBJECT_TYPE_MPEG_2_LC	This is macro AAC_OBJECT_TYPE_MPEG_2_LC.
	AAC_OBJECT_TYPE_MPEG_4_LC	This is macro AAC_OBJECT_TYPE_MPEG_4_LC.
	AAC_OBJECT_TYPE_MPEG_4_LTP	This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.
	AAC_OBJECT_TYPE_MPEG_4_SCALABLE	This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.
	AAC_SAMPLING_FREQUENCY_11025	This is macro AAC_SAMPLING_FREQUENCY_11025.
	AAC_SAMPLING_FREQUENCY_12000	This is macro AAC_SAMPLING_FREQUENCY_12000.
	AAC_SAMPLING_FREQUENCY_16000	This is macro AAC_SAMPLING_FREQUENCY_16000.
	AAC_SAMPLING_FREQUENCY_22050	This is macro AAC_SAMPLING_FREQUENCY_22050.
	AAC_SAMPLING_FREQUENCY_24000	This is macro AAC_SAMPLING_FREQUENCY_24000.
	AAC_SAMPLING_FREQUENCY_32000	This is macro AAC_SAMPLING_FREQUENCY_32000.
	AAC_SAMPLING_FREQUENCY_44100	This is macro AAC_SAMPLING_FREQUENCY_44100.
	AAC_SAMPLING_FREQUENCY_48000	This is macro AAC_SAMPLING_FREQUENCY_48000.
	AAC_SAMPLING_FREQUENCY_64000	This is macro AAC_SAMPLING_FREQUENCY_64000.
	AAC_SAMPLING_FREQUENCY_8000	This is macro AAC_SAMPLING_FREQUENCY_8000.
	AAC_SAMPLING_FREQUENCY_88200	This is macro AAC_SAMPLING_FREQUENCY_88200.
	AAC_SAMPLING_FREQUENCY_96000	This is macro AAC_SAMPLING_FREQUENCY_96000.

	AAC_SAMPLING_FREQUENCY_ALL	This is macro AAC_SAMPLING_FREQUENCY_ALL.
	AAC_VBR_NOT_SUPPORTED	This is macro AAC_VBR_NOT_SUPPORTED.
	AAC_VBR_SUPPORTED	This is macro AAC_VBR_SUPPORTED.

Structures

	Name	Description
	_bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_mpeg.h

Functions

	Name	Description
	bt_a2dp_mpeg_codec_handler	This is function bt_a2dp_mpeg_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_MPEG_H	This is macro __A2DP_CODEC_MPEG_H.
	MPEG_BITRATE_0000	This is macro MPEG_BITRATE_0000.
	MPEG_BITRATE_0001	This is macro MPEG_BITRATE_0001.
	MPEG_BITRATE_0010	This is macro MPEG_BITRATE_0010.
	MPEG_BITRATE_0011	This is macro MPEG_BITRATE_0011.
	MPEG_BITRATE_0100	This is macro MPEG_BITRATE_0100.
	MPEG_BITRATE_0101	This is macro MPEG_BITRATE_0101.
	MPEG_BITRATE_0110	This is macro MPEG_BITRATE_0110.
	MPEG_BITRATE_0111	This is macro MPEG_BITRATE_0111.
	MPEG_BITRATE_1000	This is macro MPEG_BITRATE_1000.
	MPEG_BITRATE_1001	This is macro MPEG_BITRATE_1001.
	MPEG_BITRATE_1010	This is macro MPEG_BITRATE_1010.
	MPEG_BITRATE_1011	This is macro MPEG_BITRATE_1011.
	MPEG_BITRATE_1100	This is macro MPEG_BITRATE_1100.
	MPEG_BITRATE_1101	This is macro MPEG_BITRATE_1101.
	MPEG_BITRATE_1110	This is macro MPEG_BITRATE_1110.
	MPEG_BITRATE_ALL	This is macro MPEG_BITRATE_ALL.
	MPEG_CHANNEL_MODE_ALL	This is macro MPEG_CHANNEL_MODE_ALL.
	MPEG_CHANNEL_MODE_DUAL_CHANNEL	This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.
	MPEG_CHANNEL_MODE_JOINT_STEREO	This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.
	MPEG_CHANNEL_MODE_MONO	This is macro MPEG_CHANNEL_MODE_MONO.
	MPEG_CHANNEL_MODE_STEREO	This is macro MPEG_CHANNEL_MODE_STEREO.
	MPEG_CRC_PROTECTION_NOT_SUPPORTED	This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.
	MPEG_CRC_PROTECTION_SUPPORTED	This is macro MPEG_CRC_PROTECTION_SUPPORTED.
	MPEG_LAYER_1	This is macro MPEG_LAYER_1.
	MPEG_LAYER_2	This is macro MPEG_LAYER_2.
	MPEG_LAYER_3	This is macro MPEG_LAYER_3.
	MPEG_LAYER_ALL	This is macro MPEG_LAYER_ALL.
	MPEG_MPF_1	This is macro MPEG_MPF_1.
	MPEG_MPF_2	This is macro MPEG_MPF_2.
	MPEG_SAMPLING_FREQUENCY_16000	This is macro MPEG_SAMPLING_FREQUENCY_16000.
	MPEG_SAMPLING_FREQUENCY_22050	This is macro MPEG_SAMPLING_FREQUENCY_22050.
	MPEG_SAMPLING_FREQUENCY_24000	This is macro MPEG_SAMPLING_FREQUENCY_24000.
	MPEG_SAMPLING_FREQUENCY_32000	This is macro MPEG_SAMPLING_FREQUENCY_32000.

	MPEG_SAMPLING_FREQUENCY_44100	This is macro MPEG_SAMPLING_FREQUENCY_44100.
	MPEG_SAMPLING_FREQUENCY_48000	This is macro MPEG_SAMPLING_FREQUENCY_48000.
	MPEG_SAMPLING_FREQUENCY_ALL	This is macro MPEG_SAMPLING_FREQUENCY_ALL.
	MPEG_VBR_NOT_SUPPORTED	This is macro MPEG_VBR_NOT_SUPPORTED.
	MPEG_VBR_SUPPORTED	This is macro MPEG_VBR_SUPPORTED.

Structures

	Name	Description
	_bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_sbc.h

Functions

	Name	Description
	bt_a2dp_sbc_codec_handler	This is function bt_a2dp_sbc_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_SBC_H	This is macro __A2DP_CODEC_SBC_H.
	SBC_ALLOCATION_METHOD_ALL	This is macro SBC_ALLOCATION_METHOD_ALL.
	SBC_ALLOCATION_METHOD_LOUDNESS	This is macro SBC_ALLOCATION_METHOD_LOUDNESS.
	SBC_ALLOCATION_METHOD_SNR	This is macro SBC_ALLOCATION_METHOD_SNR.
	SBC_BLOCK_LENGTH_12	This is macro SBC_BLOCK_LENGTH_12.
	SBC_BLOCK_LENGTH_16	This is macro SBC_BLOCK_LENGTH_16.
	SBC_BLOCK_LENGTH_4	This is macro SBC_BLOCK_LENGTH_4.
	SBC_BLOCK_LENGTH_8	This is macro SBC_BLOCK_LENGTH_8.
	SBC_BLOCK_LENGTH_ALL	This is macro SBC_BLOCK_LENGTH_ALL.
	SBC_CHANNEL_MODE_ALL	This is macro SBC_CHANNEL_MODE_ALL.
	SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.
	SBC_CHANNEL_MODE_JOINT_STEREO	This is macro SBC_CHANNEL_MODE_JOINT_STEREO.
	SBC_CHANNEL_MODE_MONO	This is macro SBC_CHANNEL_MODE_MONO.
	SBC_CHANNEL_MODE_STEREO	This is macro SBC_CHANNEL_MODE_STEREO.
	SBC_SAMPLING_FREQUENCY_16000	This is macro SBC_SAMPLING_FREQUENCY_16000.
	SBC_SAMPLING_FREQUENCY_32000	This is macro SBC_SAMPLING_FREQUENCY_32000.
	SBC_SAMPLING_FREQUENCY_44100	This is macro SBC_SAMPLING_FREQUENCY_44100.
	SBC_SAMPLING_FREQUENCY_48000	This is macro SBC_SAMPLING_FREQUENCY_48000.
	SBC_SAMPLING_FREQUENCY_ALL	This is macro SBC_SAMPLING_FREQUENCY_ALL.
	SBC_SUBBANDS_4	This is macro SBC_SUBBANDS_4.
	SBC_SUBBANDS_8	This is macro SBC_SUBBANDS_8.
	SBC_SUBBANDS_ALL	This is macro SBC_SUBBANDS_ALL.

Structures

	Name	Description
	_bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	_bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_private.h

Functions

	Name	Description
	_bt_a2dp_avdtp_mgr_callback	This is function _bt_a2dp_avdtp_mgr_callback.

Macros

	Name	Description
	__A2DP_PRIVATE_H	This is macro __A2DP_PRIVATE_H.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp.h

Functions

	Name	Description
	bt_avctp_cancel_command	brief Cancel a command message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_cancel_listen	brief Cancel listening for incoming connections. ingroup avctp details This function stops listening for incoming connections on the specified channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_cancel_response	brief Cancel a response message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_connect	brief Connect to a remote device. ingroup avctp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP_EVT_CHANNEL_CONNECTED or AVCTP_EVT_CONNECTION_FAILED. param channel AVCTP channel. param remote_address The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values:... more
	bt_avctp_create_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.

	bt_avctp_create_outgoing_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.
	bt_avctp_destroy_channel	brief Destroy AVCTP channel. ingroup avctp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_disconnect	brief Disconnect from a remote device. ingroup avctp details This function closes a connection to a remote device. param channel AVCTP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.
	bt_avctp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avctp details This function returns the address of the remote device associated with the channel. param channel AVCTP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avctp_get_channel_state	brief Get channel state ingroup avctp details This function return current state of the specified channel param channel AVCTP channel. return li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING
	bt_avctp_get_hci_connection	brief Get HCI connection for a channel ingroup avctp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVCTP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avctp_get_mgr	brief Return a pointer to an instance of the AVCTP manager. ingroup avctp details This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.
	bt_avctp_init	brief Initialize the AVCTP layer. ingroup avctp details This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.
	bt_avctp_listen	brief Listen for incoming connections. ingroup avctp details This function enables incoming connections on the specified AVCTP channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_send_command	brief Send a command message to a remote device. ingroup avctp details This function sends a command message to a remote device. param channel AVCTP channel. param data Message body. param data_len Message body length. param tran_id Pointer to a bt_byte where AVRCP will write transaction id assigned to the message. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_send_response	brief Send a response message to a remote device. ingroup avctp details This function sends a response message to a remote device. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. param data Message body. param data_len Message body length. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

	bt_avctp_set_callback	brief Register a AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_start	This is function bt_avctp_start.

Macros

	Name	Description
	__AVCTP_H	This is macro __AVCTP_H.
	AVCTP_CHANNEL_FLAG_LISTENING	This is macro AVCTP_CHANNEL_FLAG_LISTENING.
	AVCTP_CHANNEL_FLAG_SENDING	This is macro AVCTP_CHANNEL_FLAG_SENDING.
	AVCTP_CHANNEL_STATE_CONNECTED	This is macro AVCTP_CHANNEL_STATE_CONNECTED.
	AVCTP_CHANNEL_STATE_CONNECTING	This is macro AVCTP_CHANNEL_STATE_CONNECTING.
	AVCTP_CHANNEL_STATE_DISCONNECTING	This is macro AVCTP_CHANNEL_STATE_DISCONNECTING.
	AVCTP_CHANNEL_STATE_FREE	This is macro AVCTP_CHANNEL_STATE_FREE.
	AVCTP_CHANNEL_STATE_IDLE	This is macro AVCTP_CHANNEL_STATE_IDLE.
	AVCTP_EVT_CHANNEL_CONNECTED	< This event is generated when a channel between two AVCTP entities has been established.
	AVCTP_EVT_CHANNEL_DISCONNECTED	< This event is generated when a channel between two AVCTP entities has been terminated.
	AVCTP_EVT_COMMAND_CANCELLED	< This event is generated when a command has been canceled.
	AVCTP_EVT_COMMAND_RECEIVED	< This event is generated when a local device received a command.
	AVCTP_EVT_COMMAND_SENT	< This event is generated when a local device finished sending a command.
	AVCTP_EVT_CONNECTION_FAILED	< This event is generated when a local device failed to create a channel between two AVCTP entities.
	AVCTP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVCTP_EVT_RESPONE_RECEIVED	< This event is generated when a local device received a response.
	AVCTP_EVT_RESPONSE_CANCELLED	< This event is generated when a response has been canceled.
	AVCTP_EVT_RESPONSE_SENT	< This event is generated when a local device finished sending a response.
	AVCTP_MANAGER_STATE_IDLE	This is macro AVCTP_MANAGER_STATE_IDLE.
	AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED	This is macro AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED.
	AVCTP_TRANSPORT_FLAG_SENDING	This is macro AVCTP_TRANSPORT_FLAG_SENDING.
	AVCTP_TRANSPORT_STATE_CONNECTED	This is macro AVCTP_TRANSPORT_STATE_CONNECTED.
	AVCTP_TRANSPORT_STATE_CONNECTING	This is macro AVCTP_TRANSPORT_STATE_CONNECTING.
	AVCTP_TRANSPORT_STATE_DISCONNECTING	This is macro AVCTP_TRANSPORT_STATE_DISCONNECTING.
	AVCTP_TRANSPORT_STATE_FREE	This is macro AVCTP_TRANSPORT_STATE_FREE.
	AVCTP_TRANSPORT_STATE_IDLE	This is macro AVCTP_TRANSPORT_STATE_IDLE.

Structures

	Name	Description
	_bt_avctp_channel_t	brief AVCTP channel description ingroup avctp details This structure is used to hold information about an AVCTP channel.
	_bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.

	_bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	_bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	_bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.
	_bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.
	_bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.
	_bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	_bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	_bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.
	_bt_avctp_message_t	brief AVCTP message description ingroup avctp details This structure is used to hold information about an AVCTP message.
	_bt_avctp_mgr_t	brief AVCTP manager. ingroup avctp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avctp_get_mgr().
	_bt_avctp_transport_t	brief AVCTP transport description ingroup avctp details This structure is used to hold information about an AVCTP transport.
	bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.
	bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.
	bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.
	bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.

	bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.

Types

	Name	Description
	bt_avctp_channel_t	This is type bt_avctp_channel_t.
	bt_avctp_message_t	This is type bt_avctp_message_t.
	bt_avctp_mgr_callback_fp	brief AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with bt_avctp_set_callback()). The stack will call that function whenever a new event has been generated. param mgr AVCTP manager. param evt AVCTP event. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_mgr_t	This is type bt_avctp_mgr_t.
	bt_avctp_transport_t	This is type bt_avctp_transport_t.

Unions

	Name	Description
	_bt_avctp_event_u	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avctp_event_t	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

avctp_config.h

Macros

	Name	Description
	__AVCTP_CONFIG_H	This is macro __AVCTP_CONFIG_H.

	AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avctp_config Configuration
		• ingroup avctp
		*
		• This module describes parameters used to configure AVCTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVCTP_ALLOCATE_BUFFERS_VARS	brief Maximum number of message buffers ingroup avctp_config details This parameter defines the maximum number of buffer available for sending message.
	AVCTP_MAX_CHANNELS	brief Maximum number of AVCTP channels ingroup avctp_config details This parameter defines the maximum number of channels a local device can have with remote devices.
	AVCTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of AVCTP transports ingroup avctp_config details This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP_MAX_CHANNELS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_packet.h

Functions

	Name	Description
	_bt_avctp_packet_assembler	This is function _bt_avctp_packet_assembler.

Macros

	Name	Description
	__AVCTP_PACKET_H	This is macro __AVCTP_PACKET_H.

Structures

	Name	Description
	_bt_avctp_packet_t	This is type bt_avctp_packet_t.
	bt_avctp_packet_t	This is type bt_avctp_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_private.h

Functions

	Name	Description
	_bt_avctp_allocate_channel	This is function _bt_avctp_allocate_channel.
	_bt_avctp_allocate_message	This is function _bt_avctp_allocate_message.

	_bt_avctp_allocate_transport	This is function _bt_avctp_allocate_transport.
	_bt_avctp_find_channel	This is function _bt_avctp_find_channel.
	_bt_avctp_find_transport	This is function _bt_avctp_find_transport.
	_bt_avctp_free_channel	This is function _bt_avctp_free_channel.
	_bt_avctp_free_message	This is function _bt_avctp_free_message.
	_bt_avctp_free_transport	This is function _bt_avctp_free_transport.
	_bt_avctp_init_message_buffers	This is function _bt_avctp_init_message_buffers.
	_bt_avctp_init_signal	This is function _bt_avctp_init_signal.
	_bt_avctp_l2cap_read_data_callback	This is function _bt_avctp_l2cap_read_data_callback.
	_bt_avctp_send_ipid	This is function _bt_avctp_send_ipid.
	_bt_avctp_set_signal	This is function _bt_avctp_set_signal.
	_bt_avrcp_init_cmd_buffers	This is function _bt_avrcp_init_cmd_buffers.

Macros

	Name	Description
	__AVCTP_PRIVATE_H	This is macro __AVCTP_PRIVATE_H.

Variables

	Name	Description
	_avctp_channels	This is variable _avctp_channels.
	_avctp_max_channels	This is variable _avctp_max_channels.
	_avctp_max_message_buffers	This is variable _avctp_max_message_buffers.
	_avctp_max_rx_message_len	This is variable _avctp_max_rx_message_len.
	_avctp_max_transports	This is variable _avctp_max_transports.
	_avctp_message_buffer_headers	This is variable _avctp_message_buffer_headers.
	_avctp_message_buffers	This is variable _avctp_message_buffers.
	_avctp_rx_buffers	This is variable _avctp_rx_buffers.
	_avctp_transports	This is variable _avctp_transports.
	_ram_size_avctp_buffers	This is variable _ram_size_avctp_buffers.

Description

avdtp.h

Functions

	Name	Description
	bt_avdtp_abort_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_avdtp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more

	bt_avdtp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a AVDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up... more
	bt_avdtp_cancel_listen	brief Cancel listening for incoming connections. ingroup avdtp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_clear_media_tx_queue	brief Clear send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_avdtp_clear_media_tx_queue. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_avdtp_close_stream	brief Close a stream. ingroup avdtp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_avdtp_create_stream	brief Create a stream. ingroup avdtp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr AVDTP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_avdtp_destroy_stream	brief Destroy a stream. ingroup avdtp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_disconnect	brief Disconnect from a remote device. ingroup avdtp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_avdtp_discover	brief Discover SEPs on a remote device. ingroup avdtp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg AVDTP_EVT_DISCOVER_COMPLETED: this event is generated after last AVDTP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr AVDTP... more

	bt_avdtp_find_codec	brief Find a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	bt_avdtp_get_all_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_configuration	brief Get stream configuration. ingroup avdtp details This function requests stream configuration from a remote device. As a result of this operation the following events will be generated: @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the p ebt_para.sep_capabilities_received.caps will contain current stream configuration. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the p evt_param.get_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_stream_configuration_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual... more
	bt_avdtp_get_hci_connection	brief Get HCI connection for a stream ingroup avdtp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect. param mgr AVDTP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p strm_handle... more
	bt_avdtp_get_l2cap_channel	This is function bt_avdtp_get_l2cap_channel.
	bt_avdtp_get_mgr	brief Return a pointer to an instance of the AVDTP manager. ingroup avdtp details This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.
	bt_avdtp_get_sep	brief Get a SEP info by its ID. ingroup avdtp details This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_avdtp_register_sep. param mgr AVDTP manager. param sep_id The ID of a SEP. return li c Pointer to bt_avdtp_sep_t if the SEP is in the list of registered SEPs. li c NULL otherwise.

	bt_avdtp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup avdtp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr AVDTP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more
	bt_avdtp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup avdtp details This function returns the type of the codec currently used with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_avdtp_get_stream_config	brief Get stream's configuration. ingroup avdtp details This function returns a pointer to a structure holding the current configuration of stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_avdtp_get_stream_direction	This is function bt_avdtp_get_stream_direction.
	bt_avdtp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup avdtp details This function returns the ID of the local SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_remote_address	brief Get stream's remote BT address. ingroup avdtp details This function returns the address of the remote device associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_avdtp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup avdtp details This function returns the ID of the remote SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_state	brief Get local stream state. ingroup avdtp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr AVDTP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more
	bt_avdtp_init	brief Initialize the AVDTP layer. ingroup avdtp details This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.

	bt_avdtp_listen	brief Listen for incoming connections. ingroup avdtp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more
	bt_avdtp_open_stream	brief Open a stream. ingroup avdtp details This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.open_stream_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the p evt_param.open_stream_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function... more
	bt_avdtp_reconfigure_stream	brief Reconfigure stream. ingroup avdtp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_avdtp_register_callback	brief Register a AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote... more
	bt_avdtp_register_codec	brief Register a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	bt_avdtp_register_sep	brief Register a SEP with the local AVDTP manager. ingroup avdtp details This function is used to make a list of SEPs supported by the local ADVTP entity. param mgr AVDTP manager. param type The type of a SEP. The type can be one of the following values: @arg AVDTP_SEP_TYPE_SOURCE: The SEP is a source. @arg AVDTP_SEP_TYPE_SINK: The SEP is a sink. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_avdtp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more

	bt_avdtp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media_tx_buffer. param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more
	bt_avdtp_report_delay	brief Report delay value of a Sink to a Source. ingroup avdtp details This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the p evt_param.delay_report_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.delay_report_completed.err_code == the error code sent by the Source. param mgr AVDTP manager. param strm_handle Stream handle. param delay The delay value in 1/10 milliseconds. return li c TRUE if the function succeeds, i.e. the actual... more
	bt_avdtp_security_control	brief Exchange content protection control data. ingroup avdtp details This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_STREAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the p evt_param.security_control_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.security_control_completed.err_code == the error code sent by the remote. note The dotstack does not support content protection. Although the request can be sent it will not affect... more
	bt_avdtp_set_configuration	brief Set stream configuration. ingroup avdtp details This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the p evt_param.set_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.set_stream_configuration_completed.err_code == the error code sent by the remote and p evt_param.set_stream_configuration_completed.svc_category == the value of the first Service Category to fail. param mgr AVDTP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration. return li c TRUE if... more
	bt_avdtp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more
	bt_avdtp_start	brief Start the AVDTP layer. ingroup avdtp details This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_start_stream	brief Start a stream. ingroup avdtp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more

	bt_avdtp_suspend_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more
	bt_avdtp_unregister_codec	brief Unregister a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more

Macros

	Name	Description
	__AVDTP_H	This is macro __AVDTP_H.
	AVDTP_CODEC_OPCODE_PARSE_CONFIG	< Parse codec configuration.
	AVDTP_CODEC_OPCODE_PARSE_PACKET	This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.
	AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG	< Serialize codec configuration.
	AVDTP_CODEC_TYPE_ATRAC	< ATRAC (proprietary codec owned by Sony Corporation).
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO	< MPEG-1,2 (optional).
	AVDTP_CODEC_TYPE_MPEG2_4_AAC	< MPEG-2,4 AAC (optional, used in Apple's products).
	AVDTP_CODEC_TYPE_NON_A2DP	< Vendor specific.
	AVDTP_CODEC_TYPE_SBC	< SBC (mandatory to support in A2DP profile).
	AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T	This is macro AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T.
	AVDTP_ERROR_BAD_ACP_SEID	< The requested command indicates an invalid ACP SEP ID (not addressable)
	AVDTP_ERROR_BAD_CP_FORMAT	< The format of Content Protection Service Capability is not correct.
	AVDTP_ERROR_BAD_HEADER_FORMAT	< The request packet header format is invalid.
	AVDTP_ERROR_BAD_LENGTH	< The request packet length is not match the assumed length.
	AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT	< The format of Media Transport Capability is not correct.
	AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT	< The format of Multiplexing Service Capability is not correct.
	AVDTP_ERROR_BAD_PAYLOAD_FORMAT	< The requested command has an incorrect payload format.
	AVDTP_ERROR_BAD_RECOVERY_FORMAT	< The format of Recovery Service Capability is not correct.
	AVDTP_ERROR_BAD_RECOVERY_TYPE	< The requested Recovery Type is not defined in AVDTP.
	AVDTP_ERROR_BAD_ROHC_FORMAT	< The format of Header Compression Service Capability is not correct.
	AVDTP_ERROR_BAD_SERV_CATEGORY	< The value of Service Category in the request packet is not defined in AVDTP.
	AVDTP_ERROR_BAD_STATE	< The stream is in state that does not permit executing commands.
	AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL	< An attempt to establish a control channel has failed.
	AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT	< An attempt to establish a transport channel has failed.
	AVDTP_ERROR_INVALID_CAPABILITIES	< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.
	AVDTP_ERROR_NOT_SUPPORTED_COMMAND	< The requested command is not supported by the device.
	AVDTP_ERROR_SEP_IN_USE	< The SEP is in use.
	AVDTP_ERROR_SEP_NOT_IN_USE	< The SEP is not in use.
	AVDTP_ERROR_SUCCESS	< The operation completed with no errors.
	AVDTP_ERROR_UNSUPPORTED_CONFIGURAION	< Configuration not supported.

	AVDTP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	AVDTP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	AVDTP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	AVDTP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	AVDTP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	AVDTP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	AVDTP_EVT_DELAYREPORT_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
	AVDTP_EVT_DISCOVER_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	AVDTP_EVT_LAST	This is macro AVDTP_EVT_LAST.
	AVDTP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	AVDTP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	AVDTP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	AVDTP_EVT_NULL	This is macro AVDTP_EVT_NULL.
	AVDTP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	AVDTP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	AVDTP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	AVDTP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION	This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	AVDTP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	AVDTP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	AVDTP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.

	AVDTP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.
	AVDTP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.
	AVDTP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	AVDTP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	AVDTP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	AVDTP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	AVDTP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	AVDTP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET	This is macro AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET.
	AVDTP_MANAGER_STATE_CONNECTING	This is macro AVDTP_MANAGER_STATE_CONNECTING.
	AVDTP_MANAGER_STATE_IDLE	This is macro AVDTP_MANAGER_STATE_IDLE.
	AVDTP_MAX_STREAM_TRANSPORT_SESSION	This is macro AVDTP_MAX_STREAM_TRANSPORT_SESSION.
	AVDTP_MEDIA_TYPE_AUDIO	< Audio.
	AVDTP_MEDIA_TYPE_MULTIMEDIA	< Both Audio & Video.
	AVDTP_MEDIA_TYPE_VIDEO	< Video.
	AVDTP_SCMS_T_CP_BIT	This is macro AVDTP_SCMS_T_CP_BIT.
	AVDTP_SCMS_T_L_BIT	This is macro AVDTP_SCMS_T_L_BIT.
	AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING	< Delat reporitng.
	AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.

	AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_CAPABILITY_FLAG_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING	< Delay Reporting.
	AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_STATE_FREE	This is macro AVDTP_SEP_STATE_FREE.
	AVDTP_SEP_STATE_IDLE	This is macro AVDTP_SEP_STATE_IDLE.
	AVDTP_SEP_TYPE_SINK	< Sink (usually a device like a headphones or BMW).
	AVDTP_SEP_TYPE_SOURCE	< Source (usually a device like a phone, desktop or laptop).
	AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS	< The stream is closing transport channels.
	AVDTP_STREAM_FLAG_LISTENING	This is macro AVDTP_STREAM_FLAG_LISTENING.
	AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS	< The stream is opening transport channels.
	AVDTP_STREAM_STATE_ABORTING	< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CLOSING	< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CONFIGURED	< The stream has been configured.
	AVDTP_STREAM_STATE_IDLE	< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.
	AVDTP_STREAM_STATE_OPEN	< The stream has been opened.
	AVDTP_STREAM_STATE_STREAMING	< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.
	AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.
	AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.
	AVDTP_TRANSPORT_SESSION_TYPE_MEDIA	< Media (audio or video).
	AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY	< Recovery (currently not supported).
	AVDTP_TRANSPORT_SESSION_TYPE_REPORTING	< Reporting (currently not supported).

	bt_avdtp_connect	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more
	bt_avdtp_connect_ex	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more

Structures

	Name	Description
	_bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	_bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	_bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	_bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	_bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	_bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.

	_bt_avdtp_evt_ctrl_channel_connected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	_bt_avdtp_evt_ctrl_channel_disconnected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	_bt_avdtp_evt_ctrl_connection_failed_s	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	_bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	_bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	_bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	_bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	_bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	_bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.

	_bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	_bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	_bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	_bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	_bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	_bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	_bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

	_bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	_bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.
	_bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	_bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	_bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	_bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	_bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	_bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	_bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	_bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.
	_bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.

	_bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	_bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	_bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	_bt_avdtp_sep_t	brief SEP description ingroup avdtp details This structure is used to hold information about SEPs available on a local device.
	_bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	_bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	_bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.
	_bt_cp_header_s	This is record _bt_cp_header_s.
	_bt_media_packet_t	brief Media packet buffer ingroup avdtp details This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of ::bt_avdtp_add_media_rx_buffer and ::bt_avdtp_add_media_tx_buffer.
	bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.

	bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	bt_avdtp_evt_ctrl_channel_connected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	bt_avdtp_evt_ctrl_channel_disconnected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	bt_avdtp_evt_ctrl_connection_failed_t	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.

	bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.

	bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.
	bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.
	bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

	bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.

Types

	Name	Description
	bt_avdtp_codec_handler_fp	brief Codec handler. ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it... more
	bt_avdtp_codec_op_param_t	This is type bt_avdtp_codec_op_param_t.
	bt_avdtp_mgr_callback_fp	brief AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr AVDTP manager. param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED:... more
	bt_avdtp_sep_t	This is type bt_avdtp_sep_t.
	bt_cp_header_t	This is type bt_cp_header_t.
	bt_media_packet_t	This is type bt_media_packet_t.

Unions

	Name	Description
	_bt_avdtp_codec_op_param_u	brief Parameter to a codec handler. ingroup avdtp details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.
	_bt_avdtp_event_u	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

	bt_avdtp_event_t	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_config.h

Macros

	Name	Description
	__AVDTP_CONFIG_H	This is macro __AVDTP_CONFIG_H.
	AVDTP_ALLOCATE_BUFFERS_FUNCTION	This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

	AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avdtp_config Configuration
		• ingroup avdtp
		*
		• This module describes parameters used to configure AVDTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVDTP_ALLOCATE_BUFFERS_VARS	This is macro AVDTP_ALLOCATE_BUFFERS_VARS.
	AVDTP_CODEC_CONFIG_BUFFER_LEN	brief Size of the buffer used to store codec specific configuration. ingroup avdtp_config details Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.
	AVDTP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avdtp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP_MAX_REMOTE_DEVICES * AVDTP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.
	AVDTP_MAX_CMD_PARAM_LEN	brief Maximum length of control command parameters ingroup avdtp_config details This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP_MAX_TX_BUFFER_LEN - 2 (command header).
	AVDTP_MAX_REMOTE_DEVICES	brief Maximum number of remote devices a local device can be connected to ingroup avdtp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP_MAX_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP_MAX_REMOTE_DEVICES > AVDTP_MAX_STREAMS all memory reserved for devices in excess of AVDTP_MAX_STREAMS will be wasted. The minimum value is... more

	AVDTP_MAX_SEP	brief Maximum number of SEPs that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.
	AVDTP_MAX_STREAMS	brief Maximum number of streams that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP_MAX_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP_MAX_STREAMS > AVDTP_MAX_SEP all memory reserved for streams in excess of AVDTP_MAX_SEP will be wasted. The minimum value is 1. The maximum value is 255.
	AVDTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of transport channels. ingroup avdtp_config details Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only... more
	AVDTP_MAX_TX_BUFFER_LEN	brief Size of the transmit buffer. ingroup avdtp_config details This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP_MAX_TX_BUFFER_LEN) * AVDTP_MAX_REMOTE_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_control.h

Functions

	Name	Description
	_bt_avdtp_add_param_byte	This is function _bt_avdtp_add_param_byte.
	_bt_avdtp_add_param_uint	This is function _bt_avdtp_add_param_uint.
	_bt_avdtp_add_param_uintn	This is function _bt_avdtp_add_param_uintn.
	_bt_avdtp_control_channel_event_handler	This is function _bt_avdtp_control_channel_event_handler.
	_bt_avdtp_get_control_channel	This is function _bt_avdtp_get_control_channel.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.

Macros

	Name	Description
	__AVDTP_CONTROL_H	This is macro __AVDTP_CONTROL_H.
	AVDTP_CMD_ABORT	This is macro AVDTP_CMD_ABORT.
	AVDTP_CMD_CLOSE	This is macro AVDTP_CMD_CLOSE.
	AVDTP_CMD_DELAYREPORT	This is macro AVDTP_CMD_DELAYREPORT.
	AVDTP_CMD_DISCOVER	This is macro AVDTP_CMD_DISCOVER.
	AVDTP_CMD_GET_ALL_CAPABILITIES	This is macro AVDTP_CMD_GET_ALL_CAPABILITIES.
	AVDTP_CMD_GET_CAPABILITIES	This is macro AVDTP_CMD_GET_CAPABILITIES.
	AVDTP_CMD_GET_CONFIGURATION	This is macro AVDTP_CMD_GET_CONFIGURATION.
	AVDTP_CMD_OPEN	This is macro AVDTP_CMD_OPEN.
	AVDTP_CMD_RECONFIGURE	This is macro AVDTP_CMD_RECONFIGURE.
	AVDTP_CMD_SECURITY_CONTROL	This is macro AVDTP_CMD_SECURITY_CONTROL.
	AVDTP_CMD_SET_CONFIGURATION	This is macro AVDTP_CMD_SET_CONFIGURATION.
	AVDTP_CMD_START	This is macro AVDTP_CMD_START.
	AVDTP_CMD_SUSPEND	This is macro AVDTP_CMD_SUSPEND.

	AVDTP_CTLR_CHANNEL_EVT_CONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.
	AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_NOTHING	This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.
	AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED	This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTING	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.
	AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.
	AVDTP_CTRL_MESSAGE_TYPE_ACCEPT	This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.
	AVDTP_CTRL_MESSAGE_TYPE_COMMAND	This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.
	AVDTP_CTRL_MESSAGE_TYPE_FLD	This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.
	AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT.
	AVDTP_CTRL_MESSAGE_TYPE_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.
	AVDTP_CTRL_PACKET_TYPE_CONTINUE	This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.
	AVDTP_CTRL_PACKET_TYPE_END	This is macro AVDTP_CTRL_PACKET_TYPE_END.
	AVDTP_CTRL_PACKET_TYPE_FLD	This is macro AVDTP_CTRL_PACKET_TYPE_FLD.
	AVDTP_CTRL_PACKET_TYPE_SIGNLE	This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.
	AVDTP_CTRL_PACKET_TYPE_START	This is macro AVDTP_CTRL_PACKET_TYPE_START.

Structures

	Name	Description
	_bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	_bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	_bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.

Description

avdtp_private.h

Functions

	Name	Description
	_bt_avdtp_allocate_buffers	This is function _bt_avdtp_allocate_buffers.
	_bt_avdtp_allocate_cmd	This is function _bt_avdtp_allocate_cmd.
	_bt_avdtp_allocate_sep	This is function _bt_avdtp_allocate_sep.
	_bt_avdtp_allocate_sep_config	This is function _bt_avdtp_allocate_sep_config.
	_bt_avdtp_allocate_stream	This is function _bt_avdtp_allocate_stream.
	_bt_avdtp_allocate_transport_channel	This is function _bt_avdtp_allocate_transport_channel.
	_bt_avdtp_allocate_transport_session_id	This is function _bt_avdtp_allocate_transport_session_id.
	_bt_avdtp_begin_tc_channel_operation	This is function _bt_avdtp_begin_tc_channel_operation.
	_bt_avdtp_commit_tc_channel_operation	This is function _bt_avdtp_commit_tc_channel_operation.
	_bt_avdtp_control_channel_accept_handler	This is function _bt_avdtp_control_channel_accept_handler.
	_bt_avdtp_control_channel_cmd_handler	This is function _bt_avdtp_control_channel_cmd_handler.
	_bt_avdtp_control_channel_reject_handler	This is function _bt_avdtp_control_channel_reject_handler.
	_bt_avdtp_execute_tc_channel_operation	This is function _bt_avdtp_execute_tc_channel_operation.
	_bt_avdtp_find_listening_stream	This is function _bt_avdtp_find_listening_stream.
	_bt_avdtp_find_stream	This is function _bt_avdtp_find_stream.
	_bt_avdtp_find_stream_by_remote_sep_id	This is function _bt_avdtp_find_stream_by_remote_sep_id.
	_bt_avdtp_find_stream_by_sep_id	This is function _bt_avdtp_find_stream_by_sep_id.
	_bt_avdtp_free_cmd	This is function _bt_avdtp_free_cmd.

	_bt_avdtp_free_sep_config	This is function _bt_avdtp_free_sep_config.
	_bt_avdtp_free_stream	This is function _bt_avdtp_free_stream.
	_bt_avdtp_free_transport_channel	This is function _bt_avdtp_free_transport_channel.
	_bt_avdtp_init_cmd_buffers	This is function _bt_avdtp_init_cmd_buffers.
	_bt_avdtp_init_sep_config_buffers	This is function _bt_avdtp_init_sep_config_buffers.
	_bt_avdtp_init_signal	This is function _bt_avdtp_init_signal.
	_bt_avdtp_is_sep_inuse	This is function _bt_avdtp_is_sep_inuse.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_read_caps	This is function _bt_avdtp_read_caps.
	_bt_avdtp_register_transport_channel_for_operation	This is function _bt_avdtp_register_transport_channel_for_operation.
	_bt_avdtp_send_command	This is function _bt_avdtp_send_command.
	_bt_avdtp_send_media_packet	This is function _bt_avdtp_send_media_packet.
	_bt_avdtp_set_signal	This is function _bt_avdtp_set_signal.
	_bt_avdtp_transport_l2cap_read_data_callback	This is function _bt_avdtp_transport_l2cap_read_data_callback.
	_bt_avdtp_transport_l2cap_state_changed_callback	This is function _bt_avdtp_transport_l2cap_state_changed_callback.
	_bt_avdtp_write_caps	This is function _bt_avdtp_write_caps.

Macros

	Name	Description
	__AVDTP_PRIVATE_H	This is macro __AVDTP_PRIVATE_H.
	AVDTP_TC_OPCODE_CONNECT	This is macro AVDTP_TC_OPCODE_CONNECT.
	AVDTP_TC_OPCODE_DISCONNECT	This is macro AVDTP_TC_OPCODE_DISCONNECT.
	AVDTP_TC_STATUS_ERROR	This is macro AVDTP_TC_STATUS_ERROR.
	AVDTP_TC_STATUS_SUCCESS	This is macro AVDTP_TC_STATUS_SUCCESS.

Types

	Name	Description
	bt_avdtp_transport_op_callback_fp	This is type bt_avdtp_transport_op_callback_fp.

Variables

	Name	Description
	_avdtp_cmd_buffer_headers	This is variable _avdtp_cmd_buffer_headers.
	_avdtp_cmd_buffers	This is variable _avdtp_cmd_buffers.
	_avdtp_cmd_param_buffers	This is variable _avdtp_cmd_param_buffers.
	_avdtp_codec_cfg_buffers	This is variable _avdtp_codec_cfg_buffers.
	_avdtp_control_channels	This is variable _avdtp_control_channels.
	_avdtp_l2cap_media_packet_buffer	This is variable _avdtp_l2cap_media_packet_buffer.
	_avdtp_listen_sep_buffers	This is variable _avdtp_listen_sep_buffers.
	_avdtp_max_cmd_buffers	This is variable _avdtp_max_cmd_buffers.
	_avdtp_max_cmd_param_len	This is variable _avdtp_max_cmd_param_len.
	_avdtp_max_codec_config_buffer_len	This is variable _avdtp_max_codec_config_buffer_len.
	_avdtp_max_control_channels	This is variable _avdtp_max_control_channels.
	_avdtp_max_seps	This is variable _avdtp_max_seps.
	_avdtp_max_streams	This is variable _avdtp_max_streams.
	_avdtp_max_transport_channels	This is variable _avdtp_max_transport_channels.
	_avdtp_max_tx_buffer_len	This is variable _avdtp_max_tx_buffer_len.
	_avdtp_rcv_sep_caps	This is variable _avdtp_rcv_sep_caps.
	_avdtp_rcv_sep_codec_cfg_buffer	This is variable _avdtp_rcv_sep_codec_cfg_buffer.
	_avdtp_sep_cfg_buffer_headers	This is variable _avdtp_sep_cfg_buffer_headers.
	_avdtp_sep_cfg_buffers	This is variable _avdtp_sep_cfg_buffers.
	_avdtp_seps	This is variable _avdtp_seps.
	_avdtp_streams	This is variable _avdtp_streams.
	_avdtp_transport_channels	This is variable _avdtp_transport_channels.
	_avdtp_tx_buffers	This is variable _avdtp_tx_buffers.
	_ram_size_avdtp_buffers	This is variable _ram_size_avdtp_buffers.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avrcp.h

Functions

	Name	Description
	bt_avrcp_add_to_now_playing	brief Add to "now playing" list. ingroup avrcp details This function adds a media element specified by p element_id to the "now playing" list on the target. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be added to the "now playing" list. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_cancel_find	brief Cancel finding Targets ingroup avrcp details This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_EVT_SEARCH_COMPLETED event will be generated. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_cancel_listen	brief Cancel listening for incoming connections. ingroup avrcp details This function stops listening for incoming connections on a specified channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_connect	brief Connect to a remote device. ingroup avrcp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_EVT_CONTROL_CONNECTION_FAILED. param channel AVRCP channel. param remote_address The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE otherwise.... more
	bt_avrcp_create_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_create_outgoing_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_destroy_channel	brief Destroy AVRCP channel. ingroup avrcp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_disconnect	brief Disconnect from a remote device. ingroup avrcp details This function closes a connection to a remote device. param channel AVRCP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.
	bt_avrcp_find_controller	brief Find Controller ingroup avrcp details This function looks for a controller on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_target	brief Find Target ingroup avrcp details This function looks for a target on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_targets	brief Find Targets ingroup avrcp details This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed. param search_length The amount of time the search will be performed for. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_get_browsing_channel_state	brief Get AVCTP browsing channel state. ingroup avrcp details This function returns status of the AVCTP browsing channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING
	bt_avrcp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avrcp details This function returns the address of the remote device associated with the channel. param channel AVRCP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avrcp_get_company_id_list	brief Get Company ID list. ingroup avrcp details This function requests a list of company id's supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_control_channel_state	brief Get AVCTP control channel state. ingroup avrcp details This function returns status of the AVCTP control channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

	bt_avrcp_get_current_player_application_setting_value	brief Get current player setting values. ingroup avrcp details This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_current_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c... more
	bt_avrcp_get_element_attributes	brief Get media element attributes. ingroup avrcp details This function requests the attributes of the element specified with p element_id. note Currently p element_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track. param channel AVRCP channel. param element_id UID of the media element whose attributes are requested. param attr_mask Bitmask that defines which attributes are requested. This value can be a combination of the following values: li AVC_MEDIA_ATTR_FLAG_TITLE li AVC_MEDIA_ATTR_FLAG_ARTIST li AVC_MEDIA_ATTR_FLAG_ALBUM li AVC_MEDIA_ATTR_FLAG_NUMBER li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER li AVC_MEDIA_ATTR_FLAG_GENRE li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME li AVC_MEDIA_ATTR_FLAG_ALL return li c TRUE if... more
	bt_avrcp_get_hci_connection	brief Get HCI connection for a channel ingroup avrcp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVRCP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avrcp_get_mgr	brief Return a pointer to an instance of the AVRCP manager. ingroup avrcp details This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.
	bt_avrcp_get_play_status	brief Get playback status. ingroup avrcp details This function requests the status of the currently playing media at the target. param channel AVRCP channel. param repeat_interval Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_player_application_setting_attr_text	brief Get player setting attribute text. ingroup avrcp details This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_settings_text_t::setting_id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_text_t structure initialized as stated above. return li c... more

	bt_avrcp_get_player_application_setting_value_text	brief Get player setting value text. ingroup avrcp details This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_text_t structure initialized as stated... more
	bt_avrcp_get_supported_event_id_list	brief Get supported events. ingroup avrcp details This function requests a list of events supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_battery_status	brief Inform controller's battery status. ingroup avrcp details This function is used to inform the target about the controller's battery status. param channel AVRCP channel. param status Battery status. This can be one of the following values: li AVC_BATTERY_STATUS_NORMAL li AVC_BATTERY_STATUS_WARNING li AVC_BATTERY_STATUS_CRITICAL li AVC_BATTERY_STATUS_EXTERNAL li AVC_BATTERY_STATUS_FULL_CHARGE return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_displayable_character_set	brief Inform displayable character set. ingroup avrcp details This function informs the list of character set supported by the controller to the target. param channel AVRCP channel. param charset_list List of displayable character sets. param charset_count Number of entries in the list. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_init_controller	brief Initialize AVRCP to be used in controller mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.
	bt_avrcp_init_target	brief Initialize AVRCP to be used in target mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called. param company_id The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used. param supported_events Bitmask that specifies events supported by the target. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED li... more
	bt_avrcp_list_player_application_setting_attributes	brief Get supported player setting attributes. ingroup avrcp details This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_list_player_application_setting_values	brief Get player setting attribute values. ingroup avrcp details This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_listen	brief Listen for incoming connections. ingroup avrcp details This function enables incoming connections on the specified AVRCP channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_play_item	brief Play media item. ingroup avrcp details This function starts playing an item indicated by the UID. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be played. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_register_notification	brief Register notification. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. param channel AVRCP channel. param event_id Event Id. This value can be one of the following values: li AVC_EVENT_PLAYBACK_STATUS_CHANGED li AVC_EVENT_TRACK_CHANGED li AVC_EVENT_TRACK_REACHED_END li AVC_EVENT_TRACK_REACHED_START li AVC_EVENT_PLAYBACK_POS_CHANGED li AVC_EVENT_BATT_STATUS_CHANGED li AVC_EVENT_SYSTEM_STATUS_CHANGED li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_UIDS_CHANGED li AVC_EVENT_VOLUME_CHANGED param playback_interval The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_register_notifications	brief Register notifications. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call. note This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event. param channel AVRCP channel. param event_mask Bitmask that specifies which events to register for. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED li AVC_EVENT_FLAG_ALL return li c TRUE if the function... more

	bt_avrcp_send_cmd	brief Send AVRCP command ingroup avrcp details This function sends a command to the remote device. param channel AVRCP channel. param command Command to be sent. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_absolute_volume	brief Set absolute volume. ingroup avrcp details This function is used to set an absolute volume to be used by the rendering device. param channel AVRCP channel. param volume Volume return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_addressed_player	brief Set addressed player. ingroup avrcp details This function is used to inform the target of which media player the controller wishes to control. param channel AVRCP channel. param player_id Player Id. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_player_application_setting_value	brief Set player setting attribute values. ingroup avrcp details This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param attr_id_list List of setting attribute ids. param attr_value_list List of... more
	bt_avrcp_start	brief Start the AVRCP layer. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the c callback and c callback_param in the bt_avrcp_mgr_t structure. param mgr AVRCP manager. param callback The callback function that will be called when the AVRCP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more
	bt_avrcp_tg_send_element_attributes	brief Send media element attributes ingroup avrcp details This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status
	bt_avrcp_tg_set_absolute_volume	brief Set absolute volume ingroup avrcp details This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.
	bt_avrcp_tg_set_battery_status	brief Set battery status ingroup avrcp details This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified. param status Battery status
	bt_avrcp_tg_set_channel_absolute_volume	This is function bt_avrcp_tg_set_channel_absolute_volume.
	bt_avrcp_tg_set_current_track	brief Set current track ingroup avrcp details This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.

	bt_avrcp_tg_set_play_status	local target control * • brief Set playback status • ingroup avrcp * • details This function is used to set the playback status when AVRCP is running in target mode. • If there are active connections with controllers, the ones that registered • for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified. * • param song_length The length of the current track. • param song_position The position of the current track. • param play_status Playback status. This value can be one of the following values: • li AVC_PLAY_STATUS_STOPPED • li AVC_PLAY_STATUS_PLAYING • li AVC_PLAY_STATUS_PAUSED • li AVC_PLAY_STATUS_FW_SEEK • li AVC_PLAY_STATUS_REV_SEEK
	bt_avrcp_tg_set_system_status	brief Set system status ingroup avrcp details This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status

Macros

	Name	Description
	__AVRCP_H	This is macro __AVRCP_H.
	AVCTP_ERROR_BAD_STATE	This is macro AVCTP_ERROR_BAD_STATE.
	AVCTP_ERROR_SUCCESS	This is macro AVCTP_ERROR_SUCCESS.
	AVCTP_MESSAGE_PACKET_TYPE_CONTINUE	This is macro AVCTP_MESSAGE_PACKET_TYPE_CONTINUE.
	AVCTP_MESSAGE_PACKET_TYPE_END	This is macro AVCTP_MESSAGE_PACKET_TYPE_END.
	AVCTP_MESSAGE_PACKET_TYPE_SINGLE	This is macro AVCTP_MESSAGE_PACKET_TYPE_SINGLE.
	AVCTP_MESSAGE_PACKET_TYPE_START	This is macro AVCTP_MESSAGE_PACKET_TYPE_START.
	AVCTP_MESSAGE_TYPE_COMMAND	This is macro AVCTP_MESSAGE_TYPE_COMMAND.
	AVCTP_MESSAGE_TYPE_RESPONSE	This is macro AVCTP_MESSAGE_TYPE_RESPONSE.
	AVRCP_BTSIG_COMPANY_ID	This is macro AVRCP_BTSIG_COMPANY_ID.
	AVRCP_CHANNEL_FLAG_LISTENING	This is macro AVRCP_CHANNEL_FLAG_LISTENING.
	AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED	This is macro AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED.
	AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS	This is macro AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS.
	AVRCP_CHANNEL_FLAG_SENDING	This is macro AVRCP_CHANNEL_FLAG_SENDING.
	AVRCP_CHANNEL_STATE_CONNECTED	This is macro AVRCP_CHANNEL_STATE_CONNECTED.
	AVRCP_CHANNEL_STATE_CONNECTING	This is macro AVRCP_CHANNEL_STATE_CONNECTING.
	AVRCP_CHANNEL_STATE_DISCONNECTING	This is macro AVRCP_CHANNEL_STATE_DISCONNECTING.
	AVRCP_CHANNEL_STATE_FREE	This is macro AVRCP_CHANNEL_STATE_FREE.

	AVRCP_CHANNEL_STATE_IDLE	This is macro AVRCP_CHANNEL_STATE_IDLE.
	AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED	< This event is generated when a local device received a response to a "add to now playing" request.
	AVRCP_EVT_ADDRESSED_PLAYER_CHANGED	< This event is generated when a local device received a "addressed player changed" notification.
	AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED	< This event is generated when a local device received a "available players changed" notification.
	AVRCP_EVT_BATT_STATUS_CHANGED	< This event is generated when a local device received a "battery status changed" notification.
	AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED	< This event is generated when a local device received a "battery status of controller" command.
	AVRCP_EVT_BROWSING_CHANNEL_CONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been established.
	AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been terminated.
	AVRCP_EVT_BROWSING_CONNECTION_FAILED	< This event is generated when a local device failed to create a browsing channel between two AVRCP entities.
	AVRCP_EVT_COMPANY_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get company id" request.
	AVRCP_EVT_CONTROL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVRCP entities has been established.
	AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVRCP entities has been terminated.
	AVRCP_EVT_CONTROL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVRCP entities.
	AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED	< This event is generated when a local device received a "displayable chracter set command" request.
	AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED	< This event is generated when a local device received a "get element attributes" request.
	AVRCP_EVT_EVENT_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get supported events" request.
	AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get media element attributes" request.
	AVRCP_EVT_GET_PLAY_STATUS_RECEIVED	< This event is generated when a local device received a response to a "get play status" request.
	AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED	< This event is generated when a local device received a response to a "inform battery status" request.
	AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED	< This event is generated when a local device received a response to a "inform displayable character set" request.
	AVRCP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED	< This event is generated when a local device received a "now playing content changed" notification.
	AVRCP_EVT_PANEL_COMMAND_RECEIVED	< This event is generated when a local device received a PASS THROUGH command.
	AVRCP_EVT_PANEL_RESPONSE_RECEIVED	< This event is generated when a local device received a response to a PASS THROUGH command.
	AVRCP_EVT_PLAY_ITEM_COMPLETED	< This event is generated when a local device received a response to a "play item" request.
	AVRCP_EVT_PLAYBACK_POS_CHANGED	< This event is generated when a local device received a "playback position changed" notification.

	AVRCP_EVT_PLAYBACK_STATUS_CHANGED	< This event is generated when a local device received a "play status changed" notification.
	AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED	< This event is generated when a local device received a "player application setting changed" notification.
	AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get current player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get supported player setting attributes" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
	AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED	< This event is generated when a local device received a "register notification" request.
	AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED	< This event is generated when a local device received a response to a "register notification" request.
	AVRCP_EVT_SEARCH_COMPLETED	< This event is generated when a local device completed searching for nearby targets.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED	< This event is generated when a local device received a response to a "set absolute volume" request.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED	< This event is generated when a local device received a "set absolute volume" request.
	AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED	< This event is generated when a local device received a response to a "set addressed player" request.
	AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED	< This event is generated when a local device received a response to a "set player setting attribute values" request.
	AVRCP_EVT_SYSTEM_STATUS_CHANGED	< This event is generated when a local device received a "system status changed" notification.
	AVRCP_EVT_TRACK_CHANGED	< This event is generated when a local device received a "track changed changed" notification.
	AVRCP_EVT_TRACK_REACHED_END	< This event is generated when a local device received a "track reached end" notification.
	AVRCP_EVT_TRACK_REACHED_START	< This event is generated when a local device received a "track reached start" notification.
	AVRCP_EVT_UIDS_CHANGED	< This event is generated when a local device received a "UIDs changed" notification.
	AVRCP_EVT_VOLUME_CHANGED	< This event is generated when a local device received a "volume changed" notification.
	AVRCP_MANAGER_FLAG_SEARCHING	This is macro AVRCP_MANAGER_FLAG_SEARCHING.
	AVRCP_MANAGER_STATE_IDLE	This is macro AVRCP_MANAGER_STATE_IDLE.

Structures

	Name	Description
	_bt_avrcp_channel_t	brief AVRCP channel description ingroup avrcp details This structure is used to hold information about an AVRCP channel.
	_bt_avrcp_device_s	This is type bt_avrcp_device_t.
	_bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.

	_bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.
	_bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	_bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	_bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	_bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	_bt_avrcp_evt_search_completed_s	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.
	_bt_avrcp_mgr_t	brief AVRCP manager. ingroup avrcp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avrcp_get_mgr().
	bt_avrcp_device_t	This is type bt_avrcp_device_t.
	bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.
	bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.
	bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	bt_avrcp_evt_search_completed_t	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.

Types

	Name	Description
	bt_avrcp_channel_t	This is type bt_avrcp_channel_t.
	bt_avrcp_find_callback_fp	brief Find Controller/Target callback. ingroup avrcp details This callback is called when search for Controller/Target has finished. param found This can be one of the following values: li c TRUE if Controller/Target was found. li c FALSE otherwise. param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.
	bt_avrcp_mgr_callback_fp	brief AVRCP application callback. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated. param mgr AVRCP manager. param evt AVRCP event. The event can be one of the following values: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is... more
	bt_avrcp_mgr_t	This is type bt_avrcp_mgr_t.

Unions

	Name	Description
	_bt_avrcp_event_u	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avrcp_event_t	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

avrcp_command.h

Functions

	Name	Description
	bt_avrcp_get_subuint_info	brief Get subunit info ingroup avrcp details This function is used to request subunit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_unit_info	brief Get unit info ingroup avrcp details This function is used to request unit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_button_click	brief Send AV/C Panel Subunit "click" PASS THROUGH command ingroup avrcp details This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PANEL_BUTTON_RELEASED param channel AVRCP channel. param button_id Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_send_panel_control	brief Send AV/C Panel Subunit "control" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_CTYPE_CONTROL. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_press_panel_control	brief Send AV/C Panel Subunit "pressed" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_PRESSED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_release_panel_control	brief Send AV/C Panel Subunit "released" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_RELEASED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_simple_panel_cmd	brief Send AV/C Panel Subunit PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command to the target. param channel AVRCP channel. param ctype Command type. This value can be on of the following values: li AVC_CTYPE_CONTROL 0 li AVC_CTYPE_STATUS 1 li AVC_CTYPE_SPECIFIC_IQUIRY 2 li AVC_CTYPE_NOTIFY 3 li AVC_CTYPE_GENERAL_INQUORY 4 param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	__AVRCP_COMMAND_H	This is macro __AVRCP_COMMAND_H.
	AVC_BATTERY_STATUS_CRITICAL	This is macro AVC_BATTERY_STATUS_CRITICAL.
	AVC_BATTERY_STATUS_EXTERNAL	This is macro AVC_BATTERY_STATUS_EXTERNAL.
	AVC_BATTERY_STATUS_FULL_CHARGE	This is macro AVC_BATTERY_STATUS_FULL_CHARGE.
	AVC_BATTERY_STATUS_NORMAL	• addtogroup avrcp • @{ • @name Battery status
	AVC_BATTERY_STATUS_WARNING	This is macro AVC_BATTERY_STATUS_WARNING.
	AVC_CAPABILITY_COMPANY_ID	This is macro AVC_CAPABILITY_COMPANY_ID.
	AVC_CAPABILITY_EVENTS_SUPPORTED	This is macro AVC_CAPABILITY_EVENTS_SUPPORTED.
	AVC_CMD_GENERAL_POWER	This is macro AVC_CMD_GENERAL_POWER.
	AVC_CMD_GENERAL_RESERVE	This is macro AVC_CMD_GENERAL_RESERVE.
	AVC_CMD_GENERAL_SUBUNIT_INFO	This is macro AVC_CMD_GENERAL_SUBUNIT_INFO.
	AVC_CMD_GENERAL_UNIT_INFO	This is macro AVC_CMD_GENERAL_UNIT_INFO.
	AVC_CMD_GENERAL_VENDOR_DEPENDENT	This is macro AVC_CMD_GENERAL_VENDOR_DEPENDENT.
	AVC_CMD_GENERAL_VERSION	This is macro AVC_CMD_GENERAL_VERSION.
	AVC_CMD_PANEL_PASS_THROUGH	This is macro AVC_CMD_PANEL_PASS_THROUGH.
	AVC_CTYPE_CONTROL	• addtogroup avrcp • @{ • @name Command types
	AVC_CTYPE_GENERAL_INQUORY	This is macro AVC_CTYPE_GENERAL_INQUORY.

	AVC_CTYPE_NOTIFY	This is macro AVC_CTYPE_NOTIFY.
	AVC_CTYPE_SPECIFIC_IQUIRY	This is macro AVC_CTYPE_SPECIFIC_IQUIRY.
	AVC_CTYPE_STATUS	This is macro AVC_CTYPE_STATUS.
	AVC_EVENT_ADDRESSED_PLAYER_CHANGED	< The Addressed Player has been changed, see 6.9.2.
	AVC_EVENT_AVAILABLE_PLAYERS_CHANGED	< The available players have changed, see 6.9
	AVC_EVENT_BATT_STATUS_CHANGED	< Change in battery status
	AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED	This is macro AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED.
	AVC_EVENT_FLAG_ALL	This is macro AVC_EVENT_FLAG_ALL.
	AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED	This is macro AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED.
	AVC_EVENT_FLAG_BATT_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_BATT_STATUS_CHANGED.
	AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED	This is macro AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED	This is macro AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED	• addtogroup avrcp • @{ • @name Notifications mask
	AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED	This is macro AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED.
	AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED.
	AVC_EVENT_FLAG_TRACK_CHANGED	This is macro AVC_EVENT_FLAG_TRACK_CHANGED.
	AVC_EVENT_FLAG_TRACK_REACHED_END	This is macro AVC_EVENT_FLAG_TRACK_REACHED_END.
	AVC_EVENT_FLAG_TRACK_REACHED_START	This is macro AVC_EVENT_FLAG_TRACK_REACHED_START.
	AVC_EVENT_FLAG_UIDS_CHANGED	This is macro AVC_EVENT_FLAG_UIDS_CHANGED.
	AVC_EVENT_FLAG_VOLUME_CHANGED	This is macro AVC_EVENT_FLAG_VOLUME_CHANGED.
	AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED	< The content of the Now Playing list has changed, see 6.9.5.
	AVC_EVENT_PLAYBACK_POS_CHANGED	< Change in playback position. Returned after the specified playback notification change notification interval
	AVC_EVENT_PLAYBACK_STATUS_CHANGED	< Change in playback status of the current track.
	AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED	< Change in player application setting
	AVC_EVENT_SYSTEM_STATUS_CHANGED	< Change in system status
	AVC_EVENT_TRACK_CHANGED	< Change of current track
	AVC_EVENT_TRACK_REACHED_END	< Reached end of a track
	AVC_EVENT_TRACK_REACHED_START	< Reached start of a track
	AVC_EVENT_UIDS_CHANGED	< The UIDs have changed, see 6.10.3.3.
	AVC_EVENT_VOLUME_CHANGED	< The volume has been changed locally on the TG, see 6.13.3
	AVC_FLAG_BROWSING_CMD	This is macro AVC_FLAG_BROWSING_CMD.
	AVC_FLAG_PANEL_CLICK	This is macro AVC_FLAG_PANEL_CLICK.
	AVC_FLAG_RESPONSE	This is macro AVC_FLAG_RESPONSE.
	AVC_MAXEVENTS	This is macro AVC_MAXEVENTS.
	AVC_MEDIA_ATTR_FLAG_ALBUM	This is macro AVC_MEDIA_ATTR_FLAG_ALBUM.
	AVC_MEDIA_ATTR_FLAG_ALL	This is macro AVC_MEDIA_ATTR_FLAG_ALL.
	AVC_MEDIA_ATTR_FLAG_ARTIST	This is macro AVC_MEDIA_ATTR_FLAG_ARTIST.

	AVC_MEDIA_ATTR_FLAG_GENRE	This is macro AVC_MEDIA_ATTR_FLAG_GENRE.
	AVC_MEDIA_ATTR_FLAG_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_NUMBER.
	AVC_MEDIA_ATTR_FLAG_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_FLAG_PLAYING_TIME.
	AVC_MEDIA_ATTR_FLAG_TITLE	• addtogroup avrcp • @{ • @name Media attribute bitmask
	AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER.
	AVC_MEDIA_ATTR_ID_ALBUM	This is macro AVC_MEDIA_ATTR_ID_ALBUM.
	AVC_MEDIA_ATTR_ID_ARTIST	This is macro AVC_MEDIA_ATTR_ID_ARTIST.
	AVC_MEDIA_ATTR_ID_GENRE	This is macro AVC_MEDIA_ATTR_ID_GENRE.
	AVC_MEDIA_ATTR_ID_NUMBER	This is macro AVC_MEDIA_ATTR_ID_NUMBER.
	AVC_MEDIA_ATTR_ID_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_ID_PLAYING_TIME.
	AVC_MEDIA_ATTR_ID_TITLE	• addtogroup avrcp • @{ • @name Media attribute IDs
	AVC_MEDIA_ATTR_ID_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_ID_TOTAL_NUMBER.
	AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM	Folder Item, Media Element Item The virtual filesystem containing the Browsed media content of the browsed player
	AVC_NOW_PLAYING	Media Element Item The Now Playing list (or queue) Addressed of the addressed player
	AVC_PACKET_TYPE_CONTINUE	This is macro AVC_PACKET_TYPE_CONTINUE.
	AVC_PACKET_TYPE_END	This is macro AVC_PACKET_TYPE_END.
	AVC_PACKET_TYPE_SINGLE	This is macro AVC_PACKET_TYPE_SINGLE.
	AVC_PACKET_TYPE_START	This is macro AVC_PACKET_TYPE_START.
	AVC_PANEL_BUTTON_PRESSED	This is macro AVC_PANEL_BUTTON_PRESSED.
	AVC_PANEL_BUTTON_RELEASED	This is macro AVC_PANEL_BUTTON_RELEASED.
	AVC_PANEL_OPID_0	define AVC_PANEL_OPID_RESERVER 0x10 define AVC_PANEL_OPID_RESERVER 0x11 define AVC_PANEL_OPID_RESERVER 0x12 define AVC_PANEL_OPID_RESERVER 0x13 define AVC_PANEL_OPID_RESERVER 0x14 define AVC_PANEL_OPID_RESERVER 0x15 define AVC_PANEL_OPID_RESERVER 0x16 define AVC_PANEL_OPID_RESERVER 0x17 define AVC_PANEL_OPID_RESERVER 0x18 define AVC_PANEL_OPID_RESERVER 0x19 define AVC_PANEL_OPID_RESERVER 0x1A define AVC_PANEL_OPID_RESERVER 0x1B define AVC_PANEL_OPID_RESERVER 0x1C define AVC_PANEL_OPID_RESERVER 0x1D define AVC_PANEL_OPID_RESERVER 0x1E define AVC_PANEL_OPID_RESERVER 0x1F
	AVC_PANEL_OPID_1	This is macro AVC_PANEL_OPID_1.
	AVC_PANEL_OPID_2	This is macro AVC_PANEL_OPID_2.
	AVC_PANEL_OPID_3	This is macro AVC_PANEL_OPID_3.
	AVC_PANEL_OPID_4	This is macro AVC_PANEL_OPID_4.
	AVC_PANEL_OPID_5	This is macro AVC_PANEL_OPID_5.
	AVC_PANEL_OPID_6	This is macro AVC_PANEL_OPID_6.
	AVC_PANEL_OPID_7	This is macro AVC_PANEL_OPID_7.
	AVC_PANEL_OPID_8	This is macro AVC_PANEL_OPID_8.
	AVC_PANEL_OPID_9	This is macro AVC_PANEL_OPID_9.
	AVC_PANEL_OPID_A	This is macro AVC_PANEL_OPID_A.
	AVC_PANEL_OPID_ANGLE	define AVC_PANEL_OPID_RESERVED 0x4E define AVC_PANEL_OPID_RESERVED 0x4F
	AVC_PANEL_OPID_APPS_MENU	This is macro AVC_PANEL_OPID_APPS_MENU.
	AVC_PANEL_OPID_B	This is macro AVC_PANEL_OPID_B.

	AVC_PANEL_OPID_BACKWARD	This is macro AVC_PANEL_OPID_BACKWARD.
	AVC_PANEL_OPID_C	This is macro AVC_PANEL_OPID_C.
	AVC_PANEL_OPID_CHANNEL_DOWN	This is macro AVC_PANEL_OPID_CHANNEL_DOWN.
	AVC_PANEL_OPID_CHANNEL_UP	define AVC_PANEL_OPID_RESRVED 0x2D define AVC_PANEL_OPID_RESRVED 0x2E define AVC_PANEL_OPID_RESRVED 0x2F
	AVC_PANEL_OPID_CLEAR	This is macro AVC_PANEL_OPID_CLEAR.
	AVC_PANEL_OPID_CONTENTS_MENU	This is macro AVC_PANEL_OPID_CONTENTS_MENU.
	AVC_PANEL_OPID_D	This is macro AVC_PANEL_OPID_D.
	AVC_PANEL_OPID_DISPLAY_INFORMATION	This is macro AVC_PANEL_OPID_DISPLAY_INFORMATION.
	AVC_PANEL_OPID_DOT	This is macro AVC_PANEL_OPID_DOT.
	AVC_PANEL_OPID_DOWN	This is macro AVC_PANEL_OPID_DOWN.
	AVC_PANEL_OPID_EJECT	This is macro AVC_PANEL_OPID_EJECT.
	AVC_PANEL_OPID_ENTER	This is macro AVC_PANEL_OPID_ENTER.
	AVC_PANEL_OPID_EXIT	This is macro AVC_PANEL_OPID_EXIT.
	AVC_PANEL_OPID_F1	define AVC_PANEL_OPID_RESERVED 0x6D define AVC_PANEL_OPID_RESERVED 0x6E define AVC_PANEL_OPID_RESERVED 0x6F define AVC_PANEL_OPID_RESERVED 0x70
	AVC_PANEL_OPID_F2	This is macro AVC_PANEL_OPID_F2.
	AVC_PANEL_OPID_F3	This is macro AVC_PANEL_OPID_F3.
	AVC_PANEL_OPID_F4	This is macro AVC_PANEL_OPID_F4.
	AVC_PANEL_OPID_F5	This is macro AVC_PANEL_OPID_F5.
	AVC_PANEL_OPID_F6	This is macro AVC_PANEL_OPID_F6.
	AVC_PANEL_OPID_F7	This is macro AVC_PANEL_OPID_F7.
	AVC_PANEL_OPID_F8	This is macro AVC_PANEL_OPID_F8.
	AVC_PANEL_OPID_F9	This is macro AVC_PANEL_OPID_F9.
	AVC_PANEL_OPID_FAST_FORWARD	This is macro AVC_PANEL_OPID_FAST_FORWARD.
	AVC_PANEL_OPID_FAVORITE_MENU	This is macro AVC_PANEL_OPID_FAVORITE_MENU.
	AVC_PANEL_OPID_FORWARD	This is macro AVC_PANEL_OPID_FORWARD.
	AVC_PANEL_OPID_HELP	This is macro AVC_PANEL_OPID_HELP.
	AVC_PANEL_OPID_INPUT_SELECT	This is macro AVC_PANEL_OPID_INPUT_SELECT.
	AVC_PANEL_OPID_KEYBORD_FUNCTION	This is macro AVC_PANEL_OPID_KEYBORD_FUNCTION.
	AVC_PANEL_OPID_LEFT	This is macro AVC_PANEL_OPID_LEFT.
	AVC_PANEL_OPID_LEFT_DOWN	This is macro AVC_PANEL_OPID_LEFT_DOWN.
	AVC_PANEL_OPID_LEFT_UP	This is macro AVC_PANEL_OPID_LEFT_UP.
	AVC_PANEL_OPID_LINKED_CONTENT	This is macro AVC_PANEL_OPID_LINKED_CONTENT.
	AVC_PANEL_OPID_LIST	This is macro AVC_PANEL_OPID_LIST.
	AVC_PANEL_OPID_LIVE_TV	This is macro AVC_PANEL_OPID_LIVE_TV.
	AVC_PANEL_OPID_LOCK	This is macro AVC_PANEL_OPID_LOCK.
	AVC_PANEL_OPID_MUTE	This is macro AVC_PANEL_OPID_MUTE.
	AVC_PANEL_OPID_MUTE_FUNCTION	This is macro AVC_PANEL_OPID_MUTE_FUNCTION.
	AVC_PANEL_OPID_NEXT_DAY	This is macro AVC_PANEL_OPID_NEXT_DAY.
	AVC_PANEL_OPID_ON_DEMAND_MENU	This is macro AVC_PANEL_OPID_ON_DEMAND_MENU.
	AVC_PANEL_OPID_PAGE_DOWN	This is macro AVC_PANEL_OPID_PAGE_DOWN.
	AVC_PANEL_OPID_PAGE_UP	This is macro AVC_PANEL_OPID_PAGE_UP.
	AVC_PANEL_OPID_PAUSE	This is macro AVC_PANEL_OPID_PAUSE.
	AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION.

	AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION.
	AVC_PANEL_OPID_PIP_DOWN	This is macro AVC_PANEL_OPID_PIP_DOWN.
	AVC_PANEL_OPID_PIP_MOVE	This is macro AVC_PANEL_OPID_PIP_MOVE.
	AVC_PANEL_OPID_PIP_UP	This is macro AVC_PANEL_OPID_PIP_UP.
	AVC_PANEL_OPID_PLAY	This is macro AVC_PANEL_OPID_PLAY.
	AVC_PANEL_OPID_PLAY_FUNCTION	define AVC_PANEL_OPID_RESERVED 0x56 define AVC_PANEL_OPID_RESERVED 0x57 define AVC_PANEL_OPID_RESERVED 0x58 define AVC_PANEL_OPID_RESERVED 0x5A define AVC_PANEL_OPID_RESERVED 0x5B define AVC_PANEL_OPID_RESERVED 0x5C define AVC_PANEL_OPID_RESERVED 0x5D define AVC_PANEL_OPID_RESERVED 0x5E define AVC_PANEL_OPID_RESERVED 0x5F
	AVC_PANEL_OPID_POWER_STATE_FUNCTION	This is macro AVC_PANEL_OPID_POWER_STATE_FUNCTION.
	AVC_PANEL_OPID_POWER_TOGGLE	This is macro AVC_PANEL_OPID_POWER_TOGGLE.
	AVC_PANEL_OPID_PREVIOUS_CHANNEL	This is macro AVC_PANEL_OPID_PREVIOUS_CHANNEL.
	AVC_PANEL_OPID_PREVIOUS_DAY	This is macro AVC_PANEL_OPID_PREVIOUS_DAY.
	AVC_PANEL_OPID_RECORD	This is macro AVC_PANEL_OPID_RECORD.
	AVC_PANEL_OPID_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_RECORD_FUNCTION.
	AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION	This is macro AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION.
	AVC_PANEL_OPID_REWIND	This is macro AVC_PANEL_OPID_REWIND.
	AVC_PANEL_OPID_RF_BYPASS	This is macro AVC_PANEL_OPID_RF_BYPASS.
	AVC_PANEL_OPID_RIGHT	This is macro AVC_PANEL_OPID_RIGHT.
	AVC_PANEL_OPID_RIGHT_DOWN	This is macro AVC_PANEL_OPID_RIGHT_DOWN.
	AVC_PANEL_OPID_RIGHT_UP	This is macro AVC_PANEL_OPID_RIGHT_UP.
	AVC_PANEL_OPID_ROOT_MENU	This is macro AVC_PANEL_OPID_ROOT_MENU.
	AVC_PANEL_OPID_SELECT	• addtogroup avrcp • @{ • @name AV/C Panel PASS THROUGH operation IDs
	AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_DISK_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_DISK_FUNCTION.
	AVC_PANEL_OPID_SETUP_MENU	This is macro AVC_PANEL_OPID_SETUP_MENU.
	AVC_PANEL_OPID_SKIP	This is macro AVC_PANEL_OPID_SKIP.
	AVC_PANEL_OPID_SOUND_SELECT	This is macro AVC_PANEL_OPID_SOUND_SELECT.
	AVC_PANEL_OPID_STOP	This is macro AVC_PANEL_OPID_STOP.
	AVC_PANEL_OPID_STOP_FUNCTION	This is macro AVC_PANEL_OPID_STOP_FUNCTION.
	AVC_PANEL_OPID_SUBPICTURE	This is macro AVC_PANEL_OPID_SUBPICTURE.
	AVC_PANEL_OPID_TUNE_FUNCTION	This is macro AVC_PANEL_OPID_TUNE_FUNCTION.
	AVC_PANEL_OPID_UP	This is macro AVC_PANEL_OPID_UP.
	AVC_PANEL_OPID_VENDOR_UNIQUE	This is macro AVC_PANEL_OPID_VENDOR_UNIQUE.

	AVC_PANEL_OPID_VOLUME_DOWN	This is macro AVC_PANEL_OPID_VOLUME_DOWN.
	AVC_PANEL_OPID_VOLUME_UP	This is macro AVC_PANEL_OPID_VOLUME_UP.
	AVC_PANEL_OPID_ZOOM	This is macro AVC_PANEL_OPID_ZOOM.
	AVC_PDUID_ABORT_CONTINUING_RESPONSE	This is macro AVC_PDUID_ABORT_CONTINUING_RESPONSE.
	AVC_PDUID_ADD_TO_NOW_PLAYING	This is macro AVC_PDUID_ADD_TO_NOW_PLAYING.
	AVC_PDUID_CHANGE_PATH	This is macro AVC_PDUID_CHANGE_PATH.
	AVC_PDUID_GENERAL_REJECT	This is macro AVC_PDUID_GENERAL_REJECT.
	AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PDUID_GET_ELEMENT_ATTRIBUTES	This is macro AVC_PDUID_GET_ELEMENT_ATTRIBUTES.
	AVC_PDUID_GET_FOLDER_ITEMS	This is macro AVC_PDUID_GET_FOLDER_ITEMS.
	AVC_PDUID_GET_ITEM_ATTRIBUTES	This is macro AVC_PDUID_GET_ITEM_ATTRIBUTES.
	AVC_PDUID_GET_PLAY_STATUS	This is macro AVC_PDUID_GET_PLAY_STATUS.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT.
	AVC_PDUID_GETCAPABILITIES	This is macro AVC_PDUID_GETCAPABILITIES.
	AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT	This is macro AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT.
	AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET	This is macro AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET.
	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES.
	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES.
	AVC_PDUID_PLAY_ITEM	This is macro AVC_PDUID_PLAY_ITEM.
	AVC_PDUID_REGISTER_NOTIFICATION	This is macro AVC_PDUID_REGISTER_NOTIFICATION.
	AVC_PDUID_REQUEST_CONTINUING_RESPONSE	This is macro AVC_PDUID_REQUEST_CONTINUING_RESPONSE.
	AVC_PDUID_SEARCH	This is macro AVC_PDUID_SEARCH.
	AVC_PDUID_SET_ABSOLUTE_VOLUME	This is macro AVC_PDUID_SET_ABSOLUTE_VOLUME.
	AVC_PDUID_SET_ADDRESSED_PLAYER	This is macro AVC_PDUID_SET_ADDRESSED_PLAYER.
	AVC_PDUID_SET_BROWSED_PLAYER	This is macro AVC_PDUID_SET_BROWSED_PLAYER.
	AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PLAY_STATUS_ERROR	This is macro AVC_PLAY_STATUS_ERROR.
	AVC_PLAY_STATUS_FW_SEEK	This is macro AVC_PLAY_STATUS_FW_SEEK.
	AVC_PLAY_STATUS_PAUSED	This is macro AVC_PLAY_STATUS_PAUSED.
	AVC_PLAY_STATUS_PLAYING	This is macro AVC_PLAY_STATUS_PLAYING.
	AVC_PLAY_STATUS_REV_SEEK	This is macro AVC_PLAY_STATUS_REV_SEEK.
	AVC_PLAY_STATUS_STOPPED	• addtogroup avrcp • @{ • @name Play status

	AVC_PLAYER_SETTING_EQUALIZER_OFF	This is macro AVC_PLAYER_SETTING_EQUALIZER_OFF.
	AVC_PLAYER_SETTING_EQUALIZER_ON	This is macro AVC_PLAYER_SETTING_EQUALIZER_ON.
	AVC_PLAYER_SETTING_EQUALIZER_STATUS	This is macro AVC_PLAYER_SETTING_EQUALIZER_STATUS.
	AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS.
	AVC_PLAYER_SETTING_REPEAT_GROUP	This is macro AVC_PLAYER_SETTING_REPEAT_GROUP.
	AVC_PLAYER_SETTING_REPEAT_MODE_OFF	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_OFF.
	AVC_PLAYER_SETTING_REPEAT_MODE_STATUS	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_STATUS.
	AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK	This is macro AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK.
	AVC_PLAYER_SETTING_SCAN_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SCAN_ALL_TRACKS.
	AVC_PLAYER_SETTING_SCAN_GROUP	This is macro AVC_PLAYER_SETTING_SCAN_GROUP.
	AVC_PLAYER_SETTING_SCAN_OFF	This is macro AVC_PLAYER_SETTING_SCAN_OFF.
	AVC_PLAYER_SETTING_SCAN_STATUS	This is macro AVC_PLAYER_SETTING_SCAN_STATUS.
	AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS.
	AVC_PLAYER_SETTING_SHUFFLE_GROUP	This is macro AVC_PLAYER_SETTING_SHUFFLE_GROUP.
	AVC_PLAYER_SETTING_SHUFFLE_OFF	This is macro AVC_PLAYER_SETTING_SHUFFLE_OFF.
	AVC_PLAYER_SETTING_SHUFFLE_STATUS	This is macro AVC_PLAYER_SETTING_SHUFFLE_STATUS.
	AVC_RESPONSE_ACCEPTED	This is macro AVC_RESPONSE_ACCEPTED.
	AVC_RESPONSE_CHANGED	This is macro AVC_RESPONSE_CHANGED.
	AVC_RESPONSE_IMPLEMENTED	This is macro AVC_RESPONSE_IMPLEMENTED.
	AVC_RESPONSE_IN_TRANSITION	This is macro AVC_RESPONSE_IN_TRANSITION.
	AVC_RESPONSE_INTERIM	This is macro AVC_RESPONSE_INTERIM.
	AVC_RESPONSE_NOT_IMPLEMENTED	• addtogroup avrcp • @{ • @name Response types
	AVC_RESPONSE_REJECTED	This is macro AVC_RESPONSE_REJECTED.
	AVC_RESPONSE_STABLE	This is macro AVC_RESPONSE_STABLE.
	AVC_RESPONSE_TIMEOUT	This is macro AVC_RESPONSE_TIMEOUT.
	AVC_SCOPE_MEDIA_PLAYER_LIST	Media Player Item Contains all available media players None
	AVC_SEARCH	Media Element Item The results of a search operation Browsed on the browsed player
	AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_ID_IGNORE	This is macro AVC_SUBUNIT_ID_IGNORE.
	AVC_SUBUNIT_TYPE_AUDIO	This is macro AVC_SUBUNIT_TYPE_AUDIO.
	AVC_SUBUNIT_TYPE_BULLETIN_BOARD	This is macro AVC_SUBUNIT_TYPE_BULLETIN_BOARD.
	AVC_SUBUNIT_TYPE_CA	This is macro AVC_SUBUNIT_TYPE_CA.
	AVC_SUBUNIT_TYPE_CAMERA	This is macro AVC_SUBUNIT_TYPE_CAMERA.
	AVC_SUBUNIT_TYPE_CAMERA_STORAGE	This is macro AVC_SUBUNIT_TYPE_CAMERA_STORAGE.
	AVC_SUBUNIT_TYPE_DISC	This is macro AVC_SUBUNIT_TYPE_DISC.

	AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_TYPE_MONITOR	• addtogroup avrcp • @{ • @name Subunit types
	AVC_SUBUNIT_TYPE_PANEL	This is macro AVC_SUBUNIT_TYPE_PANEL.
	AVC_SUBUNIT_TYPE_PRINTER	This is macro AVC_SUBUNIT_TYPE_PRINTER.
	AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER	This is macro AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER.
	AVC_SUBUNIT_TYPE_TUNER	This is macro AVC_SUBUNIT_TYPE_TUNER.
	AVC_SUBUNIT_TYPE_UNIT	This is macro AVC_SUBUNIT_TYPE_UNIT.
	AVC_SUBUNIT_TYPE_VENDOR_UNIQUE	This is macro AVC_SUBUNIT_TYPE_VENDOR_UNIQUE.
	AVC_VOLUME_MAX	100
	AVC_VOLUME_MIN	0
	AVRCP_COMMAND_TYPE_BROWSING	This is macro AVRCP_COMMAND_TYPE_BROWSING.
	AVRCP_COMMAND_TYPE_CONTROL	This is macro AVRCP_COMMAND_TYPE_CONTROL.
	bt_avrcp_0_click	This is macro bt_avrcp_0_click.
	bt_avrcp_0_press	This is macro bt_avrcp_0_press.
	bt_avrcp_0_release	This is macro bt_avrcp_0_release.
	bt_avrcp_1_click	This is macro bt_avrcp_1_click.
	bt_avrcp_1_press	This is macro bt_avrcp_1_press.
	bt_avrcp_1_release	This is macro bt_avrcp_1_release.
	bt_avrcp_2_click	This is macro bt_avrcp_2_click.
	bt_avrcp_2_press	This is macro bt_avrcp_2_press.
	bt_avrcp_2_release	This is macro bt_avrcp_2_release.
	bt_avrcp_3_click	This is macro bt_avrcp_3_click.
	bt_avrcp_3_press	This is macro bt_avrcp_3_press.
	bt_avrcp_3_release	This is macro bt_avrcp_3_release.
	bt_avrcp_4_click	This is macro bt_avrcp_4_click.
	bt_avrcp_4_press	This is macro bt_avrcp_4_press.
	bt_avrcp_4_release	This is macro bt_avrcp_4_release.
	bt_avrcp_5_click	This is macro bt_avrcp_5_click.
	bt_avrcp_5_press	This is macro bt_avrcp_5_press.
	bt_avrcp_5_release	This is macro bt_avrcp_5_release.
	bt_avrcp_6_click	This is macro bt_avrcp_6_click.
	bt_avrcp_6_press	This is macro bt_avrcp_6_press.
	bt_avrcp_6_release	This is macro bt_avrcp_6_release.
	bt_avrcp_7_click	This is macro bt_avrcp_7_click.
	bt_avrcp_7_press	This is macro bt_avrcp_7_press.
	bt_avrcp_7_release	This is macro bt_avrcp_7_release.
	bt_avrcp_8_click	This is macro bt_avrcp_8_click.
	bt_avrcp_8_press	This is macro bt_avrcp_8_press.
	bt_avrcp_8_release	This is macro bt_avrcp_8_release.
	bt_avrcp_9_click	This is macro bt_avrcp_9_click.
	bt_avrcp_9_press	This is macro bt_avrcp_9_press.
	bt_avrcp_9_release	This is macro bt_avrcp_9_release.
	bt_avrcp_angle_click	This is macro bt_avrcp_angle_click.
	bt_avrcp_angle_press	This is macro bt_avrcp_angle_press.
	bt_avrcp_angle_release	This is macro bt_avrcp_angle_release.
	bt_avrcp_backward_click	This is macro bt_avrcp_backward_click.
	bt_avrcp_backward_press	This is macro bt_avrcp_backward_press.
	bt_avrcp_backward_release	This is macro bt_avrcp_backward_release.

	bt_avrcp_channel_down_click	This is macro bt_avrcp_channel_down_click.
	bt_avrcp_channel_down_press	This is macro bt_avrcp_channel_down_press.
	bt_avrcp_channel_down_release	This is macro bt_avrcp_channel_down_release.
	bt_avrcp_channel_up_click	This is macro bt_avrcp_channel_up_click.
	bt_avrcp_channel_up_press	This is macro bt_avrcp_channel_up_press.
	bt_avrcp_channel_up_release	This is macro bt_avrcp_channel_up_release.
	bt_avrcp_clear_click	This is macro bt_avrcp_clear_click.
	bt_avrcp_clear_press	This is macro bt_avrcp_clear_press.
	bt_avrcp_clear_release	This is macro bt_avrcp_clear_release.
	bt_avrcp_content_menu_click	This is macro bt_avrcp_content_menu_click.
	bt_avrcp_content_menu_release	This is macro bt_avrcp_content_menu_release.
	bt_avrcp_contents_menu_press	This is macro bt_avrcp_contents_menu_press.
	bt_avrcp_display_info_click	This is macro bt_avrcp_display_info_click.
	bt_avrcp_display_info_press	This is macro bt_avrcp_display_info_press.
	bt_avrcp_display_info_release	This is macro bt_avrcp_display_info_release.
	bt_avrcp_dot_click	This is macro bt_avrcp_dot_click.
	bt_avrcp_dot_press	This is macro bt_avrcp_dot_press.
	bt_avrcp_dot_release	This is macro bt_avrcp_dot_release.
	bt_avrcp_down_click	This is macro bt_avrcp_down_click.
	bt_avrcp_down_press	This is macro bt_avrcp_down_press.
	bt_avrcp_down_release	This is macro bt_avrcp_down_release.
	bt_avrcp_eject_click	This is macro bt_avrcp_eject_click.
	bt_avrcp_eject_press	This is macro bt_avrcp_eject_press.
	bt_avrcp_eject_release	This is macro bt_avrcp_eject_release.
	bt_avrcp_enter_click	This is macro bt_avrcp_enter_click.
	bt_avrcp_enter_press	This is macro bt_avrcp_enter_press.
	bt_avrcp_enter_release	This is macro bt_avrcp_enter_release.
	bt_avrcp_exit_click	This is macro bt_avrcp_exit_click.
	bt_avrcp_exit_press	This is macro bt_avrcp_exit_press.
	bt_avrcp_exit_release	This is macro bt_avrcp_exit_release.
	bt_avrcp_f1_click	This is macro bt_avrcp_f1_click.
	bt_avrcp_f1_press	This is macro bt_avrcp_f1_press.
	bt_avrcp_f1_release	This is macro bt_avrcp_f1_release.
	bt_avrcp_f2_click	This is macro bt_avrcp_f2_click.
	bt_avrcp_f2_press	This is macro bt_avrcp_f2_press.
	bt_avrcp_f2_release	This is macro bt_avrcp_f2_release.
	bt_avrcp_f3_click	This is macro bt_avrcp_f3_click.
	bt_avrcp_f3_press	This is macro bt_avrcp_f3_press.
	bt_avrcp_f3_release	This is macro bt_avrcp_f3_release.
	bt_avrcp_f4_click	This is macro bt_avrcp_f4_click.
	bt_avrcp_f4_press	This is macro bt_avrcp_f4_press.
	bt_avrcp_f4_release	This is macro bt_avrcp_f4_release.
	bt_avrcp_f5_click	This is macro bt_avrcp_f5_click.
	bt_avrcp_f5_press	This is macro bt_avrcp_f5_press.
	bt_avrcp_f5_release	This is macro bt_avrcp_f5_release.
	bt_avrcp_f6_click	This is macro bt_avrcp_f6_click.
	bt_avrcp_f6_press	This is macro bt_avrcp_f6_press.
	bt_avrcp_f6_release	This is macro bt_avrcp_f6_release.
	bt_avrcp_f7_click	This is macro bt_avrcp_f7_click.
	bt_avrcp_f7_press	This is macro bt_avrcp_f7_press.
	bt_avrcp_f7_release	This is macro bt_avrcp_f7_release.
	bt_avrcp_f8_click	This is macro bt_avrcp_f8_click.
	bt_avrcp_f8_press	This is macro bt_avrcp_f8_press.
	bt_avrcp_f8_release	This is macro bt_avrcp_f8_release.
	bt_avrcp_f9_click	This is macro bt_avrcp_f9_click.

	bt_avrcp_f9_press	This is macro bt_avrcp_f9_press.
	bt_avrcp_f9_release	This is macro bt_avrcp_f9_release.
	bt_avrcp_fast_forward_click	This is macro bt_avrcp_fast_forward_click.
	bt_avrcp_fast_forward_press	This is macro bt_avrcp_fast_forward_press.
	bt_avrcp_fast_forward_release	This is macro bt_avrcp_fast_forward_release.
	bt_avrcp_favorite_menu_click	This is macro bt_avrcp_favorite_menu_click.
	bt_avrcp_favorite_menu_release	This is macro bt_avrcp_favorite_menu_release.
	bt_avrcp_favorive_menu_press	This is macro bt_avrcp_favorive_menu_press.
	bt_avrcp_forward_click	This is macro bt_avrcp_forward_click.
	bt_avrcp_forward_press	This is macro bt_avrcp_forward_press.
	bt_avrcp_forward_release	This is macro bt_avrcp_forward_release.
	bt_avrcp_help_click	This is macro bt_avrcp_help_click.
	bt_avrcp_help_press	This is macro bt_avrcp_help_press.
	bt_avrcp_help_release	This is macro bt_avrcp_help_release.
	bt_avrcp_input_select_click	This is macro bt_avrcp_input_select_click.
	bt_avrcp_input_select_press	This is macro bt_avrcp_input_select_press.
	bt_avrcp_input_select_release	This is macro bt_avrcp_input_select_release.
	bt_avrcp_left_click	This is macro bt_avrcp_left_click.
	bt_avrcp_left_down_click	This is macro bt_avrcp_left_down_click.
	bt_avrcp_left_down_press	This is macro bt_avrcp_left_down_press.
	bt_avrcp_left_down_release	This is macro bt_avrcp_left_down_release.
	bt_avrcp_left_press	This is macro bt_avrcp_left_press.
	bt_avrcp_left_release	This is macro bt_avrcp_left_release.
	bt_avrcp_left_up_click	This is macro bt_avrcp_left_up_click.
	bt_avrcp_left_up_press	This is macro bt_avrcp_left_up_press.
	bt_avrcp_left_up_release	This is macro bt_avrcp_left_up_release.
	bt_avrcp_mute_click	This is macro bt_avrcp_mute_click.
	bt_avrcp_mute_press	This is macro bt_avrcp_mute_press.
	bt_avrcp_mute_release	This is macro bt_avrcp_mute_release.
	bt_avrcp_page_down_click	This is macro bt_avrcp_page_down_click.
	bt_avrcp_page_down_press	This is macro bt_avrcp_page_down_press.
	bt_avrcp_page_down_release	This is macro bt_avrcp_page_down_release.
	bt_avrcp_page_up_click	This is macro bt_avrcp_page_up_click.
	bt_avrcp_page_up_press	This is macro bt_avrcp_page_up_press.
	bt_avrcp_page_up_release	This is macro bt_avrcp_page_up_release.
	bt_avrcp_pause_click	This is macro bt_avrcp_pause_click.
	bt_avrcp_pause_press	This is macro bt_avrcp_pause_press.
	bt_avrcp_pause_release	This is macro bt_avrcp_pause_release.
	bt_avrcp_play_click	This is macro bt_avrcp_play_click.
	bt_avrcp_play_press	This is macro bt_avrcp_play_press.
	bt_avrcp_play_release	This is macro bt_avrcp_play_release.
	bt_avrcp_power_click	This is macro bt_avrcp_power_click.
	bt_avrcp_power_press	This is macro bt_avrcp_power_press.
	bt_avrcp_power_release	This is macro bt_avrcp_power_release.
	bt_avrcp_previous_channel_click	This is macro bt_avrcp_previous_channel_click.
	bt_avrcp_previous_channel_press	This is macro bt_avrcp_previous_channel_press.
	bt_avrcp_previous_channel_release	This is macro bt_avrcp_previous_channel_release.
	bt_avrcp_record_click	This is macro bt_avrcp_record_click.
	bt_avrcp_record_press	This is macro bt_avrcp_record_press.
	bt_avrcp_record_release	This is macro bt_avrcp_record_release.
	bt_avrcp_rewind_click	This is macro bt_avrcp_rewind_click.
	bt_avrcp_rewind_press	This is macro bt_avrcp_rewind_press.
	bt_avrcp_rewind_release	This is macro bt_avrcp_rewind_release.
	bt_avrcp_right_click	This is macro bt_avrcp_right_click.
	bt_avrcp_right_down_click	This is macro bt_avrcp_right_down_click.

	bt_avrcp_right_down_press	This is macro bt_avrcp_right_down_press.
	bt_avrcp_right_down_release	This is macro bt_avrcp_right_down_release.
	bt_avrcp_right_press	This is macro bt_avrcp_right_press.
	bt_avrcp_right_release	This is macro bt_avrcp_right_release.
	bt_avrcp_right_up_click	This is macro bt_avrcp_right_up_click.
	bt_avrcp_right_up_press	This is macro bt_avrcp_right_up_press.
	bt_avrcp_right_up_release	This is macro bt_avrcp_right_up_release.
	bt_avrcp_root_menu_click	This is macro bt_avrcp_root_menu_click.
	bt_avrcp_root_menu_press	This is macro bt_avrcp_root_menu_press.
	bt_avrcp_root_menu_release	This is macro bt_avrcp_root_menu_release.
	bt_avrcp_select_click	This is macro bt_avrcp_select_click.
	bt_avrcp_select_press	Panel operations
	bt_avrcp_select_release	This is macro bt_avrcp_select_release.
	bt_avrcp_setup_menu_click	This is macro bt_avrcp_setup_menu_click.
	bt_avrcp_setup_menu_press	This is macro bt_avrcp_setup_menu_press.
	bt_avrcp_setup_menu_release	This is macro bt_avrcp_setup_menu_release.
	bt_avrcp_sound_select_click	This is macro bt_avrcp_sound_select_click.
	bt_avrcp_sound_select_press	This is macro bt_avrcp_sound_select_press.
	bt_avrcp_sound_select_release	This is macro bt_avrcp_sound_select_release.
	bt_avrcp_stop_click	This is macro bt_avrcp_stop_click.
	bt_avrcp_stop_press	This is macro bt_avrcp_stop_press.
	bt_avrcp_stop_release	This is macro bt_avrcp_stop_release.
	bt_avrcp_subpicture_click	This is macro bt_avrcp_subpicture_click.
	bt_avrcp_subpicture_press	This is macro bt_avrcp_subpicture_press.
	bt_avrcp_subpicture_release	This is macro bt_avrcp_subpicture_release.
	bt_avrcp_up_click	This is macro bt_avrcp_up_click.
	bt_avrcp_up_press	This is macro bt_avrcp_up_press.
	bt_avrcp_up_release	This is macro bt_avrcp_up_release.
	bt_avrcp_volume_down_click	This is macro bt_avrcp_volume_down_click.
	bt_avrcp_volume_down_press	This is macro bt_avrcp_volume_down_press.
	bt_avrcp_volume_down_release	This is macro bt_avrcp_volume_down_release.
	bt_avrcp_volume_up_click	This is macro bt_avrcp_volume_up_click.
	bt_avrcp_volume_up_press	This is macro bt_avrcp_volume_up_press.
	bt_avrcp_volume_up_release	This is macro bt_avrcp_volume_up_release.

Structures

	Name	Description
	_bt_av_add_to_now_playing_s	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	_bt_av_battery_status_of_ct_s	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.
	_bt_av_capability_company_id_s	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.

	_bt_av_capability_event_id_s	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.
	_bt_av_command_t	This is record _bt_av_command_t.
	_bt_av_displayable_character_set_s	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	_bt_av_element_attribute_s	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	_bt_av_element_attributes_s	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	_bt_av_element_id_s	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	_bt_av_get_element_attributes_s	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	_bt_av_notification_addressed_player_changed_s	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	_bt_av_notification_app_setting_changed_s	This is type bt_av_notification_app_setting_changed_t.
	_bt_av_notification_battery_status_s	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.
	_bt_av_notification_playback_pos_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.
	_bt_av_notification_playback_status_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.

	_bt_av_notification_s	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	_bt_av_notification_system_status_changed_s	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.
	_bt_av_notification_track_changed_s	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	_bt_av_notification_uids_changed_s	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.
	_bt_av_notification_volume_changed_s	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.
	_bt_av_play_item_s	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	_bt_av_play_status_s	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.
	_bt_av_player_setting_current_values_s	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	_bt_av_player_setting_values_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.

	_bt_av_player_setting_values_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.
	_bt_av_player_settings_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.
	_bt_av_player_settings_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	_bt_av_player_text_s	This is type bt_av_player_text_t.
	_bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	_bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	_bt_av_set_absolute_volume_s	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	_bt_av_set_addressed_player_s	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.
	bt_av_add_to_now_playing_t	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	bt_av_battery_status_of_ct_t	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.
	bt_av_capability_company_id_t	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.
	bt_av_capability_event_id_t	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.

	bt_av_displayable_character_set_t	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	bt_av_element_attribute_t	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	bt_av_element_attributes_t	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	bt_av_element_id_t	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	bt_av_get_element_attributes_t	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	bt_av_notification_addressed_player_changed_t	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	bt_av_notification_app_setting_changed_t	This is type bt_av_notification_app_setting_changed_t.
	bt_av_notification_battery_status_t	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.
	bt_av_notification_playback_pos_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.
	bt_av_notification_playback_status_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.
	bt_av_notification_system_status_changed_t	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.

	bt_av_notification_t	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	bt_av_notification_track_changed_t	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	bt_av_notification_uids_changed_t	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.
	bt_av_notification_volume_changed_t	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.
	bt_av_play_item_t	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	bt_av_play_status_t	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.
	bt_av_player_setting_current_values_t	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	bt_av_player_setting_values_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.
	bt_av_player_setting_values_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.

	bt_av_player_settings_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.
	bt_av_player_settings_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	bt_av_player_text_t	This is type bt_av_player_text_t.
	bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	bt_av_set_absolute_volume_t	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	bt_av_set_addressed_player_t	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.

Types

	Name	Description
	bt_av_command_t	This is type bt_av_command_t.

Description

avrcp_config.h

Macros

	Name	Description
	__AVRCP_CONFIG_H	This is macro __AVRCP_CONFIG_H.

	AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avrcp_config Configuration
		• ingroup avrcp
		*
		• This module describes parameters used to configure AVRCP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVRCP_ALLOCATE_BUFFERS_VARS	This is macro AVRCP_ALLOCATE_BUFFERS_VARS.
	AVRCP_CMD_TIMEOUT	brief Command timeout ingroup avrcp_config details This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.
	AVRCP_MAX_CHANNELS	brief Maximum number of remote devices a local device can be connected to ingroup avrcp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.
	AVRCP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avrcp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS * AVRCP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.
	AVRCP_MAX_CMD_PARAM_LEN	brief Maximum length of command parameters ingroup avrcp_config details This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.
	AVRCP_MAX_DEVICE_NAME_LEN	brief Maximum length of device name ingroup avrcp_config details This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.
	AVRCP_MAX_SEARCH_RESULTS	brief Maximum number of devices to find ingroup avrcp_config details This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

Description

avrcp_config_event_handlers.h

Macros

	Name	Description
	__AVRCP_CONFIG_EVENT_HANDLERS_H	This is macro __AVRCP_CONFIG_EVENT_HANDLERS_H.
	AVRCP_COMMAND_HANDLER	This is macro AVRCP_COMMAND_HANDLER.
	AVRCP_COMMAND_SENT_HANDLER	This is macro AVRCP_COMMAND_SENT_HANDLER.
	AVRCP_RESPONSE_HANDLER	This is macro AVRCP_RESPONSE_HANDLER.
	AVRCP_RESPONSE_SENT_HANDLER	This is macro AVRCP_RESPONSE_SENT_HANDLER.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

avrcp_private.h

Functions

	Name	Description
	_bt_avrcp_allocate_browsing_cmd	This is function _bt_avrcp_allocate_browsing_cmd.
	_bt_avrcp_allocate_browsing_response	This is function _bt_avrcp_allocate_browsing_response.
	_bt_avrcp_allocate_bt_specific_cmd	This is function _bt_avrcp_allocate_bt_specific_cmd.
	_bt_avrcp_allocate_bt_specific_response	This is function _bt_avrcp_allocate_bt_specific_response.
	_bt_avrcp_allocate_channel	void _bt_avctp_l2cap_read_data_callback(struct _channel *pch, bt_byte* pdata, bt_int len); bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint profile_id);
	_bt_avrcp_allocate_cmd	This is function _bt_avrcp_allocate_cmd.
	_bt_avrcp_allocate_response	This is function _bt_avrcp_allocate_response.
	_bt_avrcp_allocate_simple_panel_cmd	This is function _bt_avrcp_allocate_simple_panel_cmd.
	_bt_avrcp_allocate_simple_panel_response	This is function _bt_avrcp_allocate_simple_panel_response.
	_bt_avrcp_find_channel	This is function _bt_avrcp_find_channel.
	_bt_avrcp_free_channel	This is function _bt_avrcp_free_channel.
	_bt_avrcp_free_cmd	This is function _bt_avrcp_free_cmd.
	_bt_avrcp_general_reject	This is function _bt_avrcp_general_reject.
	_bt_avrcp_get_tick_count	This is function _bt_avrcp_get_tick_count.
	_bt_avrcp_handle_add_to_now_playing	This is function _bt_avrcp_handle_add_to_now_playing.
	_bt_avrcp_handle_command	This is function _bt_avrcp_handle_command.
	_bt_avrcp_handle_command_sent	This is function _bt_avrcp_handle_command_sent.
	_bt_avrcp_handle_get_capabilities	This is function _bt_avrcp_handle_get_capabilities.
	_bt_avrcp_handle_get_current_player_application_setting_value	This is function _bt_avrcp_handle_get_current_player_application_setting_value.
	_bt_avrcp_handle_get_element_attributes	This is function _bt_avrcp_handle_get_element_attributes.
	_bt_avrcp_handle_get_play_status	This is function _bt_avrcp_handle_get_play_status.
	_bt_avrcp_handle_get_player_application_setting_attribute_text	This is function _bt_avrcp_handle_get_player_application_setting_attribute_text.
	_bt_avrcp_handle_get_player_application_setting_value_text	This is function _bt_avrcp_handle_get_player_application_setting_value_text.
	_bt_avrcp_handle_inform_battery_status_of_ct	This is function _bt_avrcp_handle_inform_battery_status_of_ct.
	_bt_avrcp_handle_inform_displayable_character_set	This is function _bt_avrcp_handle_inform_displayable_character_set.
	_bt_avrcp_handle_list_player_application_setting_attributes	This is function _bt_avrcp_handle_list_player_application_setting_attributes.
	_bt_avrcp_handle_list_player_application_setting_values	This is function _bt_avrcp_handle_list_player_application_setting_values.
	_bt_avrcp_handle_play_item	This is function _bt_avrcp_handle_play_item.
	_bt_avrcp_handle_register_notification	This is function _bt_avrcp_handle_register_notification.
	_bt_avrcp_handle_request_continuing_response	This is function _bt_avrcp_handle_request_continuing_response.
	_bt_avrcp_handle_response	This is function _bt_avrcp_handle_response.
	_bt_avrcp_handle_response_sent	This is function _bt_avrcp_handle_response_sent.
	_bt_avrcp_handle_set_absolute_volume	This is function _bt_avrcp_handle_set_absolute_volume.
	_bt_avrcp_handle_set_addressed_player	This is function _bt_avrcp_handle_set_addressed_player.
	_bt_avrcp_handle_set_player_application_setting_value	This is function _bt_avrcp_handle_set_player_application_setting_value.
	_bt_avrcp_init_signal	void _bt_avctp_init_message_buffers();

	_bt_avrcp_init_timer	This is function _bt_avrcp_init_timer.
	_bt_avrcp_register_next_notification	This is function _bt_avrcp_register_next_notification.
	_bt_avrcp_register_pending_notification	This is function _bt_avrcp_register_pending_notification.
	_bt_avrcp_send_notifications	This is function _bt_avrcp_send_notifications.
	_bt_avrcp_send_rejected_response	This is function _bt_avrcp_send_rejected_response.
	_bt_avrcp_set_signal	This is function _bt_avrcp_set_signal.
	_bt_avrcp_start_timer	This is function _bt_avrcp_start_timer.
	_bt_avrcp_tg_handle_get_capabilities	This is function _bt_avrcp_tg_handle_get_capabilities.
	_bt_avrcp_tg_handle_get_element_attributes	This is function _bt_avrcp_tg_handle_get_element_attributes.
	_bt_avrcp_tg_handle_get_play_status	This is function _bt_avrcp_tg_handle_get_play_status.
	_bt_avrcp_tg_handle_inform_battery_status_of_ct	This is function _bt_avrcp_tg_handle_inform_battery_status_of_ct.
	_bt_avrcp_tg_handle_inform_displayable_character_set	This is function _bt_avrcp_tg_handle_inform_displayable_character_set.
	_bt_avrcp_tg_handle_register_notification	This is function _bt_avrcp_tg_handle_register_notification.
	_bt_avrcp_tg_handle_set_absolute_volume	This is function _bt_avrcp_tg_handle_set_absolute_volume.
	_bt_avrcp_write_command_header	This is function _bt_avrcp_write_command_header.
	bt_avrcp_abort_continuing_response	This is function bt_avrcp_abort_continuing_response.
	bt_avrcp_avrcpt_request_continuing_response	This is function bt_avrcp_avrcpt_request_continuing_response.

Macros

	Name	Description
	__AVRCP_PRIVATE_H	This is macro __AVRCP_PRIVATE_H.
	AVRCP_MAX_ELEMENT_ATTRIBUTES	This is macro AVRCP_MAX_ELEMENT_ATTRIBUTES.

Variables

	Name	Description
	_avrcp_channels	This is variable _avrcp_channels.
	_avrcp_cmd_buffer_headers	This is variable _avrcp_cmd_buffer_headers.
	_avrcp_cmd_buffers	This is variable _avrcp_cmd_buffers.
	_avrcp_cmd_param_buffers	This is variable _avrcp_cmd_param_buffers.
	_avrcp_cmd_timeout	This is variable _avrcp_cmd_timeout.
	_avrcp_device_name_buffers	This is variable _avrcp_device_name_buffers.
	_avrcp_devices_buffer	This is variable _avrcp_devices_buffer.
	_avrcp_max_channels	This is variable _avrcp_max_channels.
	_avrcp_max_cmd_buffers	This is variable _avrcp_max_cmd_buffers.
	_avrcp_max_cmd_param_len	This is variable _avrcp_max_cmd_param_len.
	_avrcp_max_device_name_len	This is variable _avrcp_max_device_name_len.
	_avrcp_max_search_results	This is variable _avrcp_max_search_results.
	_bt_avrcp_command_handler	This is variable _bt_avrcp_command_handler.
	_bt_avrcp_command_sent_handler	This is variable _bt_avrcp_command_sent_handler.
	_bt_avrcp_response_handler	This is variable _bt_avrcp_response_handler.
	_bt_avrcp_response_sent_handler	This is variable _bt_avrcp_response_sent_handler.
	_ram_size_avrcp_buffers	This is variable _ram_size_avrcp_buffers.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

at_parser.h

Functions

	Name	Description
	bt_at_parse_fragment	This is function bt_at_parse_fragment.
	bt_at_parser_reset	This is function bt_at_parser_reset.

Macros

	Name	Description
	AT_EVT_CMD_CODE	This is macro AT_EVT_CMD_CODE.
	AT_EVT_CMD_COMPLETED	This is macro AT_EVT_CMD_COMPLETED.
	AT_EVT_CMD_PARAM	This is macro AT_EVT_CMD_PARAM.
	AT_EVT_CMD_READ_CODE	This is macro AT_EVT_CMD_READ_CODE.
	AT_EVT_ERROR	This is macro AT_EVT_ERROR.
	AT_EVT_OK	This is macro AT_EVT_OK.
	AT_EVT_RING	This is macro AT_EVT_RING.
	ATCMD_BUFFER_LEN	This is macro ATCMD_BUFFER_LEN.

Structures

	Name	Description
	_bt_at_parser_t	This is type bt_at_parser_t.
	bt_at_parser_t	This is type bt_at_parser_t.

Types

	Name	Description
	bt_at_parser_callback_pf	This is type bt_at_parser_callback_pf.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_bdaddr.h

Functions

	Name	Description
	bt_bdaddr_is_null	This is function bt_bdaddr_is_null.
	bt_bdaddrs_are_equal	This is function bt_bdaddrs_are_equal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

baseband.h

Macros

	Name	Description
	COD_MAJOR_AUDIO	This is macro COD_MAJOR_AUDIO.
	COD_MAJOR_COMPUTER	This is macro COD_MAJOR_COMPUTER.
	COD_MAJOR_HEALTH	This is macro COD_MAJOR_HEALTH.
	COD_MAJOR_IMAGING	This is macro COD_MAJOR_IMAGING.
	COD_MAJOR_MISC	This is macro COD_MAJOR_MISC.
	COD_MAJOR_NET_ACCESS_POINT	This is macro COD_MAJOR_NET_ACCESS_POINT.
	COD_MAJOR_PERIPHERAL	This is macro COD_MAJOR_PERIPHERAL.
	COD_MAJOR_PHONE	This is macro COD_MAJOR_PHONE.
	COD_MAJOR_TOY	This is macro COD_MAJOR_TOY.
	COD_MAJOR_UNCATEGORIZED	This is macro COD_MAJOR_UNCATEGORIZED.
	COD_MAJOR_WEARABLE	This is macro COD_MAJOR_WEARABLE.
	COD_MINOR_AV_CAMCORDER	This is macro COD_MINOR_AV_CAMCORDER.
	COD_MINOR_AV_CAR_AUDIO	This is macro COD_MINOR_AV_CAR_AUDIO.

	COD_MINOR_AV_GAMING	This is macro COD_MINOR_AV_GAMING.
	COD_MINOR_AV_HANDSFREE	This is macro COD_MINOR_AV_HANDSFREE.
	COD_MINOR_AV_HEADPHONES	This is macro COD_MINOR_AV_HEADPHONES.
	COD_MINOR_AV_HEADSET	This is macro COD_MINOR_AV_HEADSET.
	COD_MINOR_AV_HIFI_AUDIO	This is macro COD_MINOR_AV_HIFI_AUDIO.
	COD_MINOR_AV_LOUDSPEAKER	This is macro COD_MINOR_AV_LOUDSPEAKER.
	COD_MINOR_AV_MICROPHONE	This is macro COD_MINOR_AV_MICROPHONE.
	COD_MINOR_AV_PORTABLE_AUDIO	This is macro COD_MINOR_AV_PORTABLE_AUDIO.
	COD_MINOR_AV_RESERVED	This is macro COD_MINOR_AV_RESERVED.
	COD_MINOR_AV_RESERVERD2	This is macro COD_MINOR_AV_RESERVERD2.
	COD_MINOR_AV_SET_TOP_BOX	This is macro COD_MINOR_AV_SET_TOP_BOX.
	COD_MINOR_AV_UNCATEGORIZED	This is macro COD_MINOR_AV_UNCATEGORIZED.
	COD_MINOR_AV_VCR	This is macro COD_MINOR_AV_VCR.
	COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER	This is macro COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER.
	COD_MINOR_AV_VIDEO_CAMERA	This is macro COD_MINOR_AV_VIDEO_CAMERA.
	COD_MINOR_AV_VIDEO_CONFERENCING	This is macro COD_MINOR_AV_VIDEO_CONFERENCING.
	COD_MINOR_AV_VIDEO_MONITOR	This is macro COD_MINOR_AV_VIDEO_MONITOR.
	COD_MINOR_COMPUTER_DESKTOP	This is macro COD_MINOR_COMPUTER_DESKTOP.
	COD_MINOR_COMPUTER_HANDHELD	This is macro COD_MINOR_COMPUTER_HANDHELD.
	COD_MINOR_COMPUTER_LAPTOP	This is macro COD_MINOR_COMPUTER_LAPTOP.
	COD_MINOR_COMPUTER_PALMSIZED	This is macro COD_MINOR_COMPUTER_PALMSIZED.
	COD_MINOR_COMPUTER_SERVER	This is macro COD_MINOR_COMPUTER_SERVER.
	COD_MINOR_COMPUTER_UNCATEGORIZED	This is macro COD_MINOR_COMPUTER_UNCATEGORIZED.
	COD_MINOR_COMPUTER_WEARABLE	This is macro COD_MINOR_COMPUTER_WEARABLE.
	COD_MINOR_HEALTH_BPM	This is macro COD_MINOR_HEALTH_BPM.
	COD_MINOR_HEALTH_DATA_DISPLAY	This is macro COD_MINOR_HEALTH_DATA_DISPLAY.
	COD_MINOR_HEALTH_GLUCOSE_METER	This is macro COD_MINOR_HEALTH_GLUCOSE_METER.
	COD_MINOR_HEALTH_HEART_MONITOR	This is macro COD_MINOR_HEALTH_HEART_MONITOR.
	COD_MINOR_HEALTH_PULSE_OXIMETER	This is macro COD_MINOR_HEALTH_PULSE_OXIMETER.
	COD_MINOR_HEALTH_THERMOMETER	This is macro COD_MINOR_HEALTH_THERMOMETER.
	COD_MINOR_HEALTH_UNCATEGORIZED	This is macro COD_MINOR_HEALTH_UNCATEGORIZED.
	COD_MINOR_HEALTH_WEIGHING_SCALE	This is macro COD_MINOR_HEALTH_WEIGHING_SCALE.
	COD_MINOR_IMAGING_CAMERA	This is macro COD_MINOR_IMAGING_CAMERA.
	COD_MINOR_IMAGING_DISPLAY	This is macro COD_MINOR_IMAGING_DISPLAY.
	COD_MINOR_IMAGING_PRINTER	This is macro COD_MINOR_IMAGING_PRINTER.
	COD_MINOR_IMAGING_SCANNER	This is macro COD_MINOR_IMAGING_SCANNER.
	COD_MINOR_IMAGING_UNCATEGORIZED	This is macro COD_MINOR_IMAGING_UNCATEGORIZED.
	COD_MINOR_LAN_01_17	This is macro COD_MINOR_LAN_01_17.
	COD_MINOR_LAN_17_33	This is macro COD_MINOR_LAN_17_33.
	COD_MINOR_LAN_33_50	This is macro COD_MINOR_LAN_33_50.
	COD_MINOR_LAN_50_67	This is macro COD_MINOR_LAN_50_67.
	COD_MINOR_LAN_67_83	This is macro COD_MINOR_LAN_67_83.
	COD_MINOR_LAN_83_99	This is macro COD_MINOR_LAN_83_99.
	COD_MINOR_LAN_FULLY_AVAILABLE	This is macro COD_MINOR_LAN_FULLY_AVAILABLE.
	COD_MINOR_LAN_NO_SERVICE	This is macro COD_MINOR_LAN_NO_SERVICE.
	COD_MINOR_LAN_UNCATEGORIZED	This is macro COD_MINOR_LAN_UNCATEGORIZED.
	COD_MINOR_PERIPHERAL_CARD_READER	This is macro COD_MINOR_PERIPHERAL_CARD_READER.
	COD_MINOR_PERIPHERAL_COMBO	This is macro COD_MINOR_PERIPHERAL_COMBO.
	COD_MINOR_PERIPHERAL_DIGITIZER	This is macro COD_MINOR_PERIPHERAL_DIGITIZER.
	COD_MINOR_PERIPHERAL_GAMEPAD	This is macro COD_MINOR_PERIPHERAL_GAMEPAD.
	COD_MINOR_PERIPHERAL_JOYSTICK	This is macro COD_MINOR_PERIPHERAL_JOYSTICK.
	COD_MINOR_PERIPHERAL_KEYBOARD	This is macro COD_MINOR_PERIPHERAL_KEYBOARD.
	COD_MINOR_PERIPHERAL_MOUSE	This is macro COD_MINOR_PERIPHERAL_MOUSE.
	COD_MINOR_PERIPHERAL_OTHER	This is macro COD_MINOR_PERIPHERAL_OTHER.

	COD_MINOR_PERIPHERAL_REMOTE	This is macro COD_MINOR_PERIPHERAL_REMOTE.
	COD_MINOR_PERIPHERAL_SENSING	This is macro COD_MINOR_PERIPHERAL_SENSING.
	COD_MINOR_PERIPHERAL_UNCATEGORIZED	This is macro COD_MINOR_PERIPHERAL_UNCATEGORIZED.
	COD_MINOR_PHONE_CELLULAR	This is macro COD_MINOR_PHONE_CELLULAR.
	COD_MINOR_PHONE_CORDLESS	This is macro COD_MINOR_PHONE_CORDLESS.
	COD_MINOR_PHONE_ISDN	This is macro COD_MINOR_PHONE_ISDN.
	COD_MINOR_PHONE_SMART	This is macro COD_MINOR_PHONE_SMART.
	COD_MINOR_PHONE_UNCATEGORIZED	This is macro COD_MINOR_PHONE_UNCATEGORIZED.
	COD_MINOR_PHONE_WIREDMODEM	This is macro COD_MINOR_PHONE_WIREDMODEM.
	COD_MINOR_TOY_CONTROLLER	This is macro COD_MINOR_TOY_CONTROLLER.
	COD_MINOR_TOY_DOLL	This is macro COD_MINOR_TOY_DOLL.
	COD_MINOR_TOY_GAME	This is macro COD_MINOR_TOY_GAME.
	COD_MINOR_TOY_ROBOT	This is macro COD_MINOR_TOY_ROBOT.
	COD_MINOR_TOY_UNCATEGORIZED	This is macro COD_MINOR_TOY_UNCATEGORIZED.
	COD_MINOR_TOY_VEHICLE	This is macro COD_MINOR_TOY_VEHICLE.
	COD_MINOR_WEARABLE_GLASSES	This is macro COD_MINOR_WEARABLE_GLASSES.
	COD_MINOR_WEARABLE_HELMET	This is macro COD_MINOR_WEARABLE_HELMET.
	COD_MINOR_WEARABLE_JACKET	This is macro COD_MINOR_WEARABLE_JACKET.
	COD_MINOR_WEARABLE_PAGER	This is macro COD_MINOR_WEARABLE_PAGER.
	COD_MINOR_WEARABLE_UNCATEGORIZED	This is macro COD_MINOR_WEARABLE_UNCATEGORIZED.
	COD_MINOR_WEARABLE_WATCH	This is macro COD_MINOR_WEARABLE_WATCH.
	COS_AUDIO	This is macro COS_AUDIO.
	COS_CAPTURING	This is macro COS_CAPTURING.
	COS_INFORMATION	This is macro COS_INFORMATION.
	COS_LIMITED_DISCOVERABLE_MODE	This is macro COS_LIMITED_DISCOVERABLE_MODE.
	COS_NETWORKING	This is macro COS_NETWORKING.
	COS_OBJECTTRANSFER	This is macro COS_OBJECTTRANSFER.
	COS_POSITIONING	This is macro COS_POSITIONING.
	COS_RENDERING	This is macro COS_RENDERING.
	COS_TELEPHONY	This is macro COS_TELEPHONY.

Description

bt_config.h

bt_hcitr.h

Functions

	Name	Description
	bt_oem_recv	• Receive data. This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function. @param buffer Pointer to a buffer for the received data. The buffer must be long enough to accommodate the number of bytes specified by the par len parameter. @param len Number of bytes to receive. @param callback A callback function that must be called when the requested number of bytes have been received.
	bt_oem_send	• Send data. This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function. @param buffer Pointer to the data to be sent . @param len Number of bytes to send. @param callback A callback function that must be called when all data have been sent.

Types

	Name	Description
	bt_oem_recv_callback_fp	• Receive callback. This callback function is called when a receive operation initiated by bt_oem_recv() has completed. @param len Number of received bytes. The value of this parameter should always be the same as the number of bytes requested in a call to bt_oem_recv().
	bt_oem_send_callback_fp	• Send callback. This callback function is called when a send operation initiated by bt_oem_send() has completed.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_log.h

Functions

	Name	Description
	bt_log_bdaddr	This is function bt_log_bdaddr.
	bt_log_int	This is function bt_log_int.
	bt_log_linkkey	This is function bt_log_linkkey.
	bt_log_memory	This is function bt_log_memory.
	bt_log_msg	This is function bt_log_msg.
	bt_oem_log_write	brief Output log message. ingroup log details DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.

Macros

	Name	Description
	BT_LOG	This is macro BT_LOG.
	BT_LOG_EX	This is macro BT_LOG_EX.

	BT_LOG_LEVEL_A2DP_PAKCET	This is macro BT_LOG_LEVEL_A2DP_PAKCET.
	BT_LOG_LEVEL_ALL	This is macro BT_LOG_LEVEL_ALL.
	BT_LOG_LEVEL_DEBUG	This is macro BT_LOG_LEVEL_DEBUG.
	BT_LOG_LEVEL_ERROR	This is macro BT_LOG_LEVEL_ERROR.
	BT_LOG_LEVEL_INFO	This is macro BT_LOG_LEVEL_INFO.
	BT_LOG_LEVEL_OFF	This is macro BT_LOG_LEVEL_OFF.
	BT_LOGADDR	This is macro BT_LOGADDR.
	BT_LOGADDR_EX	This is macro BT_LOGADDR_EX.
	BT_LOGINT	This is macro BT_LOGINT.
	BT_LOGINT_EX	This is macro BT_LOGINT_EX.
	BT_LOGLINKKEY	This is macro BT_LOGLINKKEY.
	BT_LOGLINKKEY_EX	This is macro BT_LOGLINKKEY_EX.
	BT_LOGMEMORY	This is macro BT_LOGMEMORY.
	BT_LOGMEMORY_EX	This is macro BT_LOGMEMORY_EX.
	BT_LOGWRITE	This is macro BT_LOGWRITE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_oem.h

Functions

	Name	Description
	bt_oem_assert	This is function bt_oem_assert.
	bt_oem_get_device_class	This is function bt_oem_get_device_class.
	bt_oem_get_device_name	This is function bt_oem_get_device_name.
	bt_oem_get_pin_code	This is function bt_oem_get_pin_code.
	bt_oem_linkkey_notification	This is function bt_oem_linkkey_notification.
	bt_oem_linkkey_request	This is function bt_oem_linkkey_request.
	bt_oem_schedule_signals	This is function bt_oem_schedule_signals.
	bt_oem_ssp_callback	This is function bt_oem_ssp_callback.

Structures

	Name	Description
	_bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	_bt_linkkey_request_t	This is type bt_linkkey_request_t.
	bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	bt_linkkey_request_t	This is type bt_linkkey_request_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_oem_config.h

Functions

	Name	Description
	ATT_ALLOCATE_BUFFERS	This is function ATT_ALLOCATE_BUFFERS.
	ATT_CLIENT_ALLOCATE_BUFFERS	This is function ATT_CLIENT_ALLOCATE_BUFFERS.

	FTP_ALLOCATE_BUFFERS	This is function FTP_ALLOCATE_BUFFERS.
	GATT_CLIENT_ALLOCATE_BUFFERS	This is function GATT_CLIENT_ALLOCATE_BUFFERS.
	HCI_ALLOCATE_BUFFERS	This is function HCI_ALLOCATE_BUFFERS.
	HCITR_BCSP_ALLOCATE_BUFFERS	This is function HCITR_BCSP_ALLOCATE_BUFFERS.
	HCRP_ALLOCATE_BUFFERS	This is function HCRP_ALLOCATE_BUFFERS.
	HFP_ALLOCATE_BUFFERS	This is function HFP_ALLOCATE_BUFFERS.
	HID_ALLOCATE_BUFFERS	This is function HID_ALLOCATE_BUFFERS.
	HSP_AG_ALLOCATE_BUFFERS	This is function HSP_AG_ALLOCATE_BUFFERS.
	HSP_ALLOCATE_BUFFERS	This is function HSP_ALLOCATE_BUFFERS.
	IAP_ALLOCATE_BUFFERS	This is function IAP_ALLOCATE_BUFFERS.
	IAP_BT_ALLOCATE_BUFFERS	This is function IAP_BT_ALLOCATE_BUFFERS.
	IAP2_ALLOCATE_BUFFERS	This is function IAP2_ALLOCATE_BUFFERS.
	IAPEA_ALLOCATE_BUFFERS	This is function IAPEA_ALLOCATE_BUFFERS.
	L2CAP_ALLOCATE_BUFFERS	This is function L2CAP_ALLOCATE_BUFFERS.
	MAP_ALLOCATE_BUFFERS	This is function MAP_ALLOCATE_BUFFERS.
	OBEX_ALLOCATE_BUFFERS	This is function OBEX_ALLOCATE_BUFFERS.
	PBAP_ALLOCATE_BUFFERS	This is function PBAP_ALLOCATE_BUFFERS.
	RFCOMM_ALLOCATE_BUFFERS	This is function RFCOMM_ALLOCATE_BUFFERS.
	SDP_ALLOCATE_BUFFERS	This is function SDP_ALLOCATE_BUFFERS.
	SMP_ALLOCATE_BUFFERS	This is function SMP_ALLOCATE_BUFFERS.
	SPP_ALLOCATE_BUFFERS	This is function SPP_ALLOCATE_BUFFERS.

Macros

	Name	Description
	__BT_APP_CONFIG_H	This is macro __BT_APP_CONFIG_H.
	BT_ENABLE_BLE	This is macro BT_ENABLE_BLE.
	BT_INCLUDE_IAP	This is macro BT_INCLUDE_IAP.
	BT_INCLUDE_IAP2	This is macro BT_INCLUDE_IAP2.
	BT_INCLUDE_RFCOMM	This is macro BT_INCLUDE_RFCOMM.
	BT_LOG_LEVEL_MAX	This is macro BT_LOG_LEVEL_MAX.
	BT_LOG_LEVEL_MIN	This is macro BT_LOG_LEVEL_MIN.
	IAP_BT_MAX_TRANSPORTS	This is macro IAP_BT_MAX_TRANSPORTS.
	IAP_MAX_SESSIONS	This is macro IAP_MAX_SESSIONS.
	IAP_RX_BUFFER_SIZE	This is macro IAP_RX_BUFFER_SIZE.
	IAP2_MAX_PACKET_SIZE	This is macro IAP2_MAX_PACKET_SIZE.
	IAP2_MAX_SESSIONS	This is macro IAP2_MAX_SESSIONS.

Variables

	Name	Description
	_bt_log_level_max	This is variable _bt_log_level_max.
	_bt_log_level_min	This is variable _bt_log_level_min.

Description

bt_private.h

Macros

	Name	Description
	ARG_NOT_USED	This is macro ARG_NOT_USED.
	BT_ASSERT	This is macro BT_ASSERT.

	FALSE	This is macro FALSE.
	LOG	This is macro LOG.
	LOG_EX	This is macro LOG_EX.
	LOGADDR	This is macro LOGADDR.
	LOGADDR_EX	This is macro LOGADDR_EX.
	LOGCLEAR	This is macro LOGCLEAR.
	LOGINT	This is macro LOGINT.
	LOGINT_EX	This is macro LOGINT_EX.
	LOGMEMORY	This is macro LOGMEMORY.
	LOGMEMORY_EX	This is macro LOGMEMORY_EX.
	LOGWRITE	This is macro LOGWRITE.
	NULL	This is macro NULL.
	TRUE	This is macro TRUE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_signal.h

Functions

	Name	Description
	bt_signal_init	This is function bt_signal_init.
	bt_signal_process_pending	This is function bt_signal_process_pending.
	bt_signal_register	This is function bt_signal_register.
	bt_signal_set	This is function bt_signal_set.
	bt_signal_unregister	This is function bt_signal_unregister.

Structures

	Name	Description
	_bt_signal_t	This is record _bt_signal_t.

Types

	Name	Description
	bt_signal_handler_fp	This is type bt_signal_handler_fp.
	bt_signal_t	This is type bt_signal_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_std.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_storage.h

Functions

	Name	Description
	bt_oem_storage_get_capacity	brief Get non-volatile storage capacity. ingroup stg details Implementation of this function must return the capacity of its non-volatile storage.
	bt_oem_storage_read	brief Read from the non-volatile storage. ingroup stg details This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the c len parameter is read, the application must call the callback function specified by the c callback parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees... more
	bt_oem_storage_start	brief Begin a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt_oem_storage_stop().
	bt_oem_storage_stop	brief End a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.
	bt_oem_storage_write	brief Write to non-volatile storage. ingroup stg details This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the c callback parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not... more

Macros

	Name	Description
	__BLUETOOTH_STG_H	This is macro __BLUETOOTH_STG_H.
	CDS_LAST_DEVICE_ADDR	This is macro CDS_LAST_DEVICE_ADDR.
	CDS_SIGNATURE	This is macro CDS_SIGNATURE.
	CDS_SIGNATURE_ADDR	This is macro CDS_SIGNATURE_ADDR.
	HCI_BDADDR_LEN	This is macro HCI_BDADDR_LEN.
	HIDS_LAST_DEVICE_ADDR	This is macro HIDS_LAST_DEVICE_ADDR.
	HIDS_SIGNATURE	This is macro HIDS_SIGNATURE.
	HIDS_SIGNATURE_ADDR	This is macro HIDS_SIGNATURE_ADDR.
	LKS_FIRST_KEY_ADDR	This is macro LKS_FIRST_KEY_ADDR.
	LKS_MAX_LINK_KEYS	This is macro LKS_MAX_LINK_KEYS.
	LKS_SIGNATURE	This is macro LKS_SIGNATURE.
	LKS_SIGNATURE_ADDR	This is macro LKS_SIGNATURE_ADDR.

Types

	Name	Description
	bt_storage_callback_fp	brief Storage callback. details This callback is called when a non-volatile storage operation completes.

Description

bt_system.h

Functions

	Name	Description
	bt_sys_get_connectable	This is function bt_sys_get_connectable.
	bt_sys_get_discoverable	This is function bt_sys_get_discoverable.
	bt_sys_get_l2cap_manager	• Get the L2CAP manager. This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence. @return The L2CAP manager.
	bt_sys_init	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). This function essentially calls bt_sys_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.
	bt_sys_init_ex	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). Also, the caller must provide an SDP database. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of... more
	bt_sys_set_modes	This is function bt_sys_set_modes.
	bt_sys_start	• Start the Bluetooth system. After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCI controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function. Also, the caller must provide an SDP database. @param discoverable defines whether the device is discoverable after reset. @param connectable defines whether the device is connectable after reset. @param sdp_db SDP database data. @param sdp_db_len Length of SDP database... more

Types

	Name	Description
	bt_sys_callback_fp	• System start callback. This callback function is called when system start initiated by bt_sys_start() has completed. @param success Success of the operation: c BT_TRUE if successfull, c BT_FALSE otherwise. @param param Callback parameter that was specified when bt_sys_start() was called.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_timer.h

Enumerations

	Name	Description
	_bt_timer_id_enum	This is type bt_timer_id.

	bt_timer_id	This is type bt_timer_id.

Functions

	Name	Description
	bt_oem_timer_clear	brief Clear timer. details This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback. param timerId ID of the timer to clear.
	bt_oem_timer_set	brief Set timer. details This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately. param timerId ID of the timer to set. param milliseconds Timer interval in milliseconds param callback Timer expiration callback function.

Types

	Name	Description
	bt_timer_callback_fp	brief Timer callback. details This callback is called when a timer expires.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_types.h

Macros

	Name	Description
	BT_FALSE	This is macro BT_FALSE.
	BT_LINKKEY_LENGTH	This is macro BT_LINKKEY_LENGTH.
	BT_MAKE_BDADDR	This is macro BT_MAKE_BDADDR.
	BT_MAKE_BDADDR_LE	This is macro BT_MAKE_BDADDR_LE.
	BT_NO	This is macro BT_NO.
	BT_TRUE	This is macro BT_TRUE.
	BT_YES	This is macro BT_YES.

Structures

	Name	Description
	_bt_bdaddr_s	This is type bt_bdaddr_t.
	_bt_linkkey_t	This is type bt_linkkey_t.
	_bt_uuid_s	This is type bt_uuid_t.
	bt_bdaddr_t	This is type bt_bdaddr_t.
	bt_linkkey_t	This is type bt_linkkey_t.
	bt_uuid_t	This is type bt_uuid_t.

Types

	Name	Description
	bt_bdaddr_cp	This is type bt_bdaddr_cp.
	bt_bdaddr_p	This is type bt_bdaddr_p.
	bt_bool	This is type bt_bool.
	bt_byte_cp	This is type bt_byte_cp.
	bt_byte_p	This is type bt_byte_p.
	bt_char	This is type bt_char.

	bt_char_cp	This is type bt_char_cp.
	bt_char_p	This is type bt_char_p.
	bt_id	This is type bt_id.
	bt_int_cp	This is type bt_int_cp.
	bt_int_p	This is type bt_int_p.
	bt_linkkey_cp	This is type bt_linkkey_cp.
	bt_linkkey_p	This is type bt_linkkey_p.
	bt_long_cp	This is type bt_long_cp.
	bt_long_p	This is type bt_long_p.
	bt_uint_cp	This is type bt_uint_cp.
	bt_uint_p	This is type bt_uint_p.
	bt_ulong_cp	This is type bt_ulong_cp.
	bt_ulong_p	This is type bt_ulong_p.
	bt_uuid_cp	This is type bt_uuid_cp.
	bt_uuid_p	This is type bt_uuid_p.
	bt_uuid16	This is type bt_uuid16.
	bt_uuid32	This is type bt_uuid32.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

buffer.h

Functions

	Name	Description
	bt_alloc_buffer	This is function bt_alloc_buffer.
	bt_free_buffer	This is function bt_free_buffer.
	bt_get_buffer	This is function bt_get_buffer.
	bt_get_buffer_header	This is function bt_get_buffer_header.
	bt_get_buffer_index	This is function bt_get_buffer_index.
	bt_init_buffer_mgr	This is function bt_init_buffer_mgr.

Macros

	Name	Description
	BUFFER_HDR_LEN	This is macro BUFFER_HDR_LEN.
	BUFFER_STATE_FREE	for offsetof
	BUFFER_STATE_USED	This is macro BUFFER_STATE_USED.

Structures

	Name	Description
	_bt_buffer_header_t	This is type bt_buffer_header_t.
	_bt_buffer_mgr_t	This is type bt_buffer_mgr_t.
	bt_bt_buffer_header_p	This is type bt_bt_buffer_header_p.
	bt_buffer_header_t	This is type bt_buffer_header_t.
	bt_buffer_mgr_p	This is type bt_buffer_mgr_p.
	bt_buffer_mgr_t	This is type bt_buffer_mgr_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with

a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bufferutils.h

Functions

	Name	Description
	_bt_memcpy	This is function _bt_memcpy.
	_compact_buffer	This is function _compact_buffer.
	_expand_buffer	This is function _expand_buffer.
	_is_empty_str	This is function _is_empty_str.
	_read_bdaddr	This is function _read_bdaddr.
	_readb	This is function _readb.
	_readi	This is function _readi.
	_readin	This is function _readin.
	_readl	This is function _readl.
	_readln	This is function _readln.
	_readui	This is function _readui.
	_readuin	This is function _readuin.
	_readul	This is function _readul.
	_readuln	This is function _readuln.
	_str2ulong	This is function _str2ulong.
	_str2ulong_dec	This is function _str2ulong_dec.
	_to_lower_case	This is function _to_lower_case.
	_ulong2str	This is function _ulong2str.
	_ulong2str_dec	This is function _ulong2str_dec.
	_write_bdaddr	This is function _write_bdaddr.
	_writeb	This is function _writeb.
	_writei	This is function _writei.
	_writein	This is function _writein.
	_writel	This is function _writel.
	_writeln	This is function _writeln.
	_writes	This is function _writes.
	_writesx	This is function _writesx.
	_zero_memory	This is function _zero_memory.

Macros

Name	Description
_readbn	This is macro _readbn.
_readui	This is macro _readui.
_readuin	This is macro _readuin.
_readul	This is macro _readul.
_readuln	This is macro _readuln.
_writebn	This is macro _writebn.
bt_max	This is macro bt_max.
bt_min	This is macro bt_min.
max	This is macro max.
min	This is macro min.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

channel.h

Functions

	Name	Description
	bt_l2cap_read_data	This is function bt_l2cap_read_data.
	bt_l2cap_send_cmd	This is function bt_l2cap_send_cmd.
	bt_l2cap_send_data	brief Send data over an L2CAP channel ingroup l2cap details This function sends data over the specified L2CAP channel. param channel The L2CAP channel to send data over. param data The pointer to the data. param len The length of the data. param callback The callback function that is called when sending the data has been completed. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_send_smart_data	This is function bt_l2cap_send_smart_data.

Macros

	Name	Description
	CID_ATT	This is macro CID_ATT.
	CID_LE_SIG	This is macro CID_LE_SIG.
	CID_MAX	This is macro CID_MAX.
	CID_MAX_FIXED	This is macro CID_MAX_FIXED.
	CID_NULL	This is macro CID_NULL.
	CID_RECV	This is macro CID_RECV.
	CID_SIG	This is macro CID_SIG.
	CID_SM	This is macro CID_SM.
	CMODE_BASIC	This is macro CMODE_BASIC.
	CMODE_ERETR	This is macro CMODE_ERETR.
	CMODE_FLOW	This is macro CMODE_FLOW.
	CMODE_RETR	This is macro CMODE_RETR.
	CMODE_STRM	This is macro CMODE_STRM.
	CSTATE_CLOSED	This is macro CSTATE_CLOSED.
	CSTATE_FREE	This is macro CSTATE_FREE.
	CSTATE_OPEN	This is macro CSTATE_OPEN.
	CSTATE_WAIT_CONFIG	This is macro CSTATE_WAIT_CONFIG.
	CSTATE_WAIT_CONFIG_REQ	This is macro CSTATE_WAIT_CONFIG_REQ.
	CSTATE_WAIT_CONFIG_RSP	This is macro CSTATE_WAIT_CONFIG_RSP.
	CSTATE_WAIT_CONNECT	This is macro CSTATE_WAIT_CONNECT.
	CSTATE_WAIT_CONNECT_RSP	This is macro CSTATE_WAIT_CONNECT_RSP.
	CSTATE_WAIT_DISCONNECT	This is macro CSTATE_WAIT_DISCONNECT.
	CTYPE_CL	connectionless bt_l2cap_channel
	CTYPE_CO	connection-oriented bt_l2cap_channel
	L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT	This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.
	L2CAP_CHANNEL_FLAG_INCOMING	This is macro L2CAP_CHANNEL_FLAG_INCOMING.
	L2CAP_CHANNEL_FLAG_SENDING	This is macro L2CAP_CHANNEL_FLAG_SENDING.
	L2CAP_ERETR_RECV_STATE_RECV	This is macro L2CAP_ERETR_RECV_STATE_RECV.
	L2CAP_ERETR_RECV_STATE_REJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.
	L2CAP_ERETR_RECV_STATE_SREJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.
	L2CAP_ERETR_XMIT_STATE_WAIT_ACK	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.
	L2CAP_ERETR_XMIT_STATE_WAIT_F	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.
	L2CAP_ERETR_XMIT_STATE_XMIT	This is macro L2CAP_ERETR_XMIT_STATE_XMIT.

Structures

	Name	Description
	_bt_l2cap_channel_ext_t	This is record _bt_l2cap_channel_ext_t.
	_bt_l2cap_channel_t	This is record _bt_l2cap_channel_t.

	_bt_l2cap_frame_desc_s	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	bt_l2cap_frame_desc_t	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;

Types

	Name	Description
	bt_l2cap_channel_ext_t	This is type bt_l2cap_channel_ext_t.
	bt_l2cap_channel_t	This is type bt_l2cap_channel_t.
	bt_l2cap_read_data_callback_fp	This is type bt_l2cap_read_data_callback_fp.
	bt_l2cap_send_data_callback_fp	This is type bt_l2cap_send_data_callback_fp.
	bt_l2cap_state_changed_callback_fp	This is type bt_l2cap_state_changed_callback_fp.

Description

chmanager.h

Functions

	Name	Description
	bt_l2cap_allocate_channel	This is function bt_l2cap_allocate_channel.
	bt_l2cap_free_channel	This is function bt_l2cap_free_channel.
	bt_l2cap_get_channel	This is function bt_l2cap_get_channel.
	bt_l2cap_get_channel_by_bdaddr_cid	This is function bt_l2cap_get_channel_by_bdaddr_cid.
	bt_l2cap_get_channel_by_hconn_cid	This is function bt_l2cap_get_channel_by_hconn_cid.
	bt_l2cap_get_channel_by_hconn_dest_cid	This is function bt_l2cap_get_channel_by_hconn_dest_cid.
	bt_l2cap_get_channel_by_psm	This is function bt_l2cap_get_channel_by_psm.
	bt_l2cap_init_channels	This is function bt_l2cap_init_channels.

Description

cmdbuffer.h

Functions

	Name	Description
	bt_l2cap_alloc_cmd_buffer	This is function bt_l2cap_alloc_cmd_buffer.
	bt_l2cap_alloc_cmd_config_req	This is function bt_l2cap_alloc_cmd_config_req.
	bt_l2cap_alloc_cmd_config_res	This is function bt_l2cap_alloc_cmd_config_res.
	bt_l2cap_alloc_cmd_conn_param_update_req	This is function bt_l2cap_alloc_cmd_conn_param_update_req.
	bt_l2cap_alloc_cmd_conn_param_update_res	This is function bt_l2cap_alloc_cmd_conn_param_update_res.
	bt_l2cap_alloc_cmd_connection_req	This is function bt_l2cap_alloc_cmd_connection_req.
	bt_l2cap_alloc_cmd_connection_res	This is function bt_l2cap_alloc_cmd_connection_res.
	bt_l2cap_alloc_cmd_disconnection_req	This is function bt_l2cap_alloc_cmd_disconnection_req.
	bt_l2cap_alloc_cmd_disconnection_res	This is function bt_l2cap_alloc_cmd_disconnection_res.

	bt_l2cap_alloc_cmd_echo_req	This is function bt_l2cap_alloc_cmd_echo_req.
	bt_l2cap_alloc_cmd_echo_res	This is function bt_l2cap_alloc_cmd_echo_res.
	bt_l2cap_alloc_cmd_info_req	This is function bt_l2cap_alloc_cmd_info_req.
	bt_l2cap_alloc_cmd_info_res	This is function bt_l2cap_alloc_cmd_info_res.
	bt_l2cap_alloc_cmd_reject	This is function bt_l2cap_alloc_cmd_reject.
	bt_l2cap_free_cmd_buffer	This is function bt_l2cap_free_cmd_buffer.
	bt_l2cap_init_cmd_buffers	This is function bt_l2cap_init_cmd_buffers.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

command.h

Macros

	Name	Description
	L2CAP_CMD_CONFIG_REQUEST	This is macro L2CAP_CMD_CONFIG_REQUEST.
	L2CAP_CMD_CONFIG_RESPONSE	This is macro L2CAP_CMD_CONFIG_RESPONSE.
	L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST.
	L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE.
	L2CAP_CMD_CONN_REQUEST	This is macro L2CAP_CMD_CONN_REQUEST.
	L2CAP_CMD_CONN_RESPONSE	This is macro L2CAP_CMD_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_CMD_REJECT	This is macro L2CAP_CMD_DATA_LEN_CMD_REJECT.
	L2CAP_CMD_DATA_LEN_CONFIG_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONFIG_REQUEST.
	L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE.
	L2CAP_CMD_DATA_LEN_CONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONN_REQUEST.
	L2CAP_CMD_DATA_LEN_CONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_DCONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_DCONN_REQUEST.
	L2CAP_CMD_DATA_LEN_DCONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_DCONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_ECHO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_ECHO_REQUEST.
	L2CAP_CMD_DATA_LEN_ECHO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_ECHO_RESPONSE.
	L2CAP_CMD_DATA_LEN_INFO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_INFO_REQUEST.
	L2CAP_CMD_DATA_LEN_INFO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_INFO_RESPONSE.
	L2CAP_CMD_DCONN_REQUEST	This is macro L2CAP_CMD_DCONN_REQUEST.
	L2CAP_CMD_DCONN_RESPONSE	This is macro L2CAP_CMD_DCONN_RESPONSE.
	L2CAP_CMD_ECHO_REQUEST	This is macro L2CAP_CMD_ECHO_REQUEST.
	L2CAP_CMD_ECHO_RESPONSE	This is macro L2CAP_CMD_ECHO_RESPONSE.
	L2CAP_CMD_HEADER_LEN	This is macro L2CAP_CMD_HEADER_LEN.
	L2CAP_CMD_INFO_REQUEST	This is macro L2CAP_CMD_INFO_REQUEST.
	L2CAP_CMD_INFO_RESPONSE	This is macro L2CAP_CMD_INFO_RESPONSE.
	L2CAP_CMD_LAST	This is macro L2CAP_CMD_LAST.
	L2CAP_CMD_REJECT	This is macro L2CAP_CMD_REJECT.
	L2CAP_CMD_RESERVED	This is macro L2CAP_CMD_RESERVED.
	L2CAP_CMD_STATUS_BEING_SENT	This is macro L2CAP_CMD_STATUS_BEING_SENT.
	L2CAP_CMD_STATUS_PENDING	This is macro L2CAP_CMD_STATUS_PENDING.

	L2CAP_CMD_STATUS_WAITING_RESPONSE	This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.
	L2CAP_CONFIG_RESULT_REJECTED	This is macro L2CAP_CONFIG_RESULT_REJECTED.
	L2CAP_CONFIG_RESULT_SUCCESS	This is macro L2CAP_CONFIG_RESULT_SUCCESS.
	L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER	This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.
	L2CAP_CONFIG_RESULT_UNKNOWN_OPTION	This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.
	L2CAP_CONN_REQ_RESULT_INVALID_PSM	This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.
	L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID	This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.
	L2CAP_CONN_REQ_RESULT_NO_RESOURCES	This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.
	L2CAP_CONN_REQ_RESULT_PENDING	This is macro L2CAP_CONN_REQ_RESULT_PENDING.
	L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK	This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.
	L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED	This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.
	L2CAP_CONN_REQ_RESULT_SUCCESS	This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.
	L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.
	L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.
	L2CAP_CONN_REQ_STATUS_NO_INFO	This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.
	L2CAP_ECHO_MAX_DATA_LEN	echo request and response
	L2CAP_EXT_BI_QOS	This is macro L2CAP_EXT_BI_QOS.
	L2CAP_EXT_ENHANCED_RETRANSMISSION	This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.
	L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR	This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.
	L2CAP_EXT_EXT_WINDOW_SIZE	This is macro L2CAP_EXT_EXT_WINDOW_SIZE.
	L2CAP_EXT_FCS_OPTION	This is macro L2CAP_EXT_FCS_OPTION.
	L2CAP_EXT_FIXED_CHANNELS	This is macro L2CAP_EXT_FIXED_CHANNELS.
	L2CAP_EXT_FLOW_CONTROL	This is macro L2CAP_EXT_FLOW_CONTROL.
	L2CAP_EXT_RETRANSMISSION	This is macro L2CAP_EXT_RETRANSMISSION.
	L2CAP_EXT_STREAMING	This is macro L2CAP_EXT_STREAMING.
	L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT	This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.
	L2CAP_INFO_NOT_SUPPORTED	REDFLAG: should it be non zero?
	L2CAP_INFO_RESULT_SUCCESS	This is macro L2CAP_INFO_RESULT_SUCCESS.
	L2CAP_INFO_TYPE_CONNECTIONLESS_MTU	information request
	L2CAP_INFO_TYPE_EXTENDED_SUPPORT	This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.
	L2CAP_INFO_TYPE_FIXED_CHANNELS	This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.
	L2CAP_MAX_OPTIONS	configuration request/response
	L2CAP_OPTION_LEN_FLASH_QOS	This is macro L2CAP_OPTION_LEN_FLASH_QOS.
	L2CAP_OPTION_LEN_FLASH_RFC	This is macro L2CAP_OPTION_LEN_FLASH_RFC.
	L2CAP_OPTION_LEN_FLASH_TIMEOUT	This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.
	L2CAP_OPTION_LEN_MAX_MTU	This is macro L2CAP_OPTION_LEN_MAX_MTU.
	L2CAP_OPTION_LEN_UNKNOWN	This is macro L2CAP_OPTION_LEN_UNKNOWN.
	L2CAP_OPTION_LEN_UNKNOWN_DATA	21
	L2CAP_OPTION_TYPE_FLASH_QOS	This is macro L2CAP_OPTION_TYPE_FLASH_QOS.
	L2CAP_OPTION_TYPE_FLASH_QOS_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.
	L2CAP_OPTION_TYPE_FLASH_RFC	This is macro L2CAP_OPTION_TYPE_FLASH_RFC.
	L2CAP_OPTION_TYPE_FLASH_RFC_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.

	L2CAP_OPTION_TYPE_FLASH_TIMEOUT	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.
	L2CAP_OPTION_TYPE_MAX_MTU	This is macro L2CAP_OPTION_TYPE_MAX_MTU.
	L2CAP_OPTION_TYPE_MAX_MTU_FLAG	This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.
	L2CAP_OPTION_TYPE_UNKNOWN_FLAG	This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.
	L2CAP_REJECT_REASON_INVALID_CHANNEL	This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.
	L2CAP_REJECT_REASON_MTU_EXCEEDED	This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.
	L2CAP_REJECT_REASON_NOT_UNDERSTOOD	This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.
	L2CAP_RFC_BASIC	This is macro L2CAP_RFC_BASIC.
	L2CAP_RFC_ENHANCED_RETRANSMISSION	This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.
	L2CAP_RFC_ERETR_TIMEOUT	3 secs
	L2CAP_RFC_ERETR_TX_WINDOW	This is macro L2CAP_RFC_ERETR_TX_WINDOW.
	L2CAP_RFC_FLOW_CONTROL	This is macro L2CAP_RFC_FLOW_CONTROL.
	L2CAP_RFC_MONITOR_TIMEOUT	12 secs
	L2CAP_RFC_RETRANSMISSION	This is macro L2CAP_RFC_RETRANSMISSION.
	L2CAP_RFC_STREAMING	This is macro L2CAP_RFC_STREAMING.
	L2CAP_SERVICE_TYPE_BEST_EFFORT	This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.
	L2CAP_SERVICE_TYPE_GUARANTEED	This is macro L2CAP_SERVICE_TYPE_GUARANTEED.
	L2CAP_SERVICE_TYPE_NO_TRAFFIC	This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.

Structures

	Name	Description
	_bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	_bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	_bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	_bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	_bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	_bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	_bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	_bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	_bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	_bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	_bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	_bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	_bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	_bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	_bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	_bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	_bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	_bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	_bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	_cmd_disconnection_res	This is type pcmd_disconnection_res.
	bt_l2cap_cfg_option_p	This is type bt_l2cap_cfg_option_p.
	bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	bt_l2cap_cmd_config_req_p	This is type bt_l2cap_cmd_config_req_p.
	bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	bt_l2cap_cmd_config_res_p	This is type bt_l2cap_cmd_config_res_p.
	bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	bt_l2cap_cmd_connection_req_p	This is type bt_l2cap_cmd_connection_req_p.

	bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	bt_l2cap_cmd_connection_res_p	This is type bt_l2cap_cmd_connection_res_p.
	bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	bt_l2cap_cmd_disconnection_req_p	This is type bt_l2cap_cmd_disconnection_req_p.
	bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	bt_l2cap_cmd_echo_req_p	This is type bt_l2cap_cmd_echo_req_p.
	bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	bt_l2cap_cmd_echo_res_p	This is type bt_l2cap_cmd_echo_res_p.
	bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	bt_l2cap_cmd_header_p	This is type bt_l2cap_cmd_header_p.
	bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	bt_l2cap_cmd_info_req_p	This is type bt_l2cap_cmd_info_req_p.
	bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	bt_l2cap_cmd_info_res_p	This is type bt_l2cap_cmd_info_res_p.
	bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	bt_l2cap_cmd_reject_p	This is type bt_l2cap_cmd_reject_p.
	bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	cmd_disconnection_res	This is type cmd_disconnection_res.
	pcmd_disconnection_res	This is type pcmd_disconnection_res.

Types

	Name	Description
	pf_l2cap_cmd_callback	This is type pf_l2cap_cmd_callback.

Unions

	Name	Description
	_bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	_bt_l2cap_command_t	This is type bt_l2cap_command_t.
	bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	bt_l2cap_command_t	This is type bt_l2cap_command_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

config.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC.

Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

csr.h

Functions

	Name	Description
	btx_csr_alloc_bccmd_getreq	This is function btx_csr_alloc_bccmd_getreq.
	btx_csr_alloc_bccmd_setreq	This is function btx_csr_alloc_bccmd_setreq.

	btx_csr_autobaud	brief Configure controller's UART speed. ingroup btx_csr details This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.
	btx_csr_bc7_sel_host_interface_h4	brief Configure controller's UART speed and host interface. ingroup btx_csr details This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HOST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.
	btx_csr_enable_tx	brief Enable/disable transmitter ingroup btx_csr param enable Specifies whether the transmitter should be enable or disabled. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_exec_hq_script	This is function btx_csr_exec_hq_script.
	btx_csr_exec_script	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. The c script must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the c callback with the first parameter TRUE. Otherwise the c callback is called with the first parameter FALSE. If support for multiple firmware revisions is neede use btx_csr_patch_controller(). param script Array of... more
	btx_csr_get_cached_temperature	brief Get chip's cached temperature ingroup btx_csr param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_ps_var	This is function btx_csr_get_ps_var.
	btx_csr_get_ps_var_ex	This is function btx_csr_get_ps_var_ex.
	btx_csr_get_rssi_acl	brief Get RSSI ingroup btx_csr details This function retrieves the RSSI for a given HCI ACL handle. param hconn ACL connection handle. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_script__dsp_script__PB_109_DSP_rev8	brief Return script for patching DSP in CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_101_CSR8811_CSP28_UART	brief Return script for patching CSR8x11 A06 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_109_CSR8811_REV16	brief Return script for patching CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_173_CSR8X11_REV1	brief Return script for patching CSR8x11 A12 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_27_R20_BC6ROM_A04	brief Return script for patching BlueCore 6 ingroup btx_csr
	btx_csr_get_script__PB_90_REV6	brief Return script for patching CSR8810 (BlueCore 7) ingroup btx_csr
	btx_csr_get_var	This is function btx_csr_get_var.
	btx_csr_init	brief Initialize CSR support layer. ingroup btx_csr details This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.
	btx_csr_init_hq_script	This is function btx_csr_init_hq_script.

	btx_csr_patch_controller	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. Each entry of the c scripts array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the c scripts. If there is a matching script it is loaded to the controller and c callback is called with the first parameter TRUE. If no suitable script found c callback is called with the first parameter FALSE. param scripts Array of patch scripts.... more
	btx_csr_register_bccmd_listener	This is function btx_csr_register_bccmd_listener.
	btx_csr_send_dsp_config_data	This is function btx_csr_send_dsp_config_data.
	btx_csr_send_next_hq_script_packet	This is function btx_csr_send_next_hq_script_packet.
	btx_csr_set_ps_var	This is function btx_csr_set_ps_var.
	btx_csr_set_ps_var_ex	This is function btx_csr_set_ps_var_ex.
	btx_csr_set_ps_vars	brief Write PS variables ingroup btx_csr details param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_ps_vars_ex	brief Write PS variables ingroup btx_csr param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param store param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_var	This is function btx_csr_set_var.
	btx_csr_unregister_bccmd_listener	This is function btx_csr_unregister_bccmd_listener.
	btx_csr_warm_reset	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.
	btx_csr_warm_reset_ex	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the c callback is called right after the command packet has been transmitted to the controller. param callback The callback function that will be called after the warm reset command has been sent to the controller. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more

Macros

	Name	Description
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN.
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION.
	CSR_I2S_STREAM_CFG_KEY_MASTER	This is macro CSR_I2S_STREAM_CFG_KEY_MASTER.
	CSR_I2S_STREAM_CFG_KEY_MCLK	This is macro CSR_I2S_STREAM_CFG_KEY_MCLK.
	CSR_I2S_STREAM_CFG_KEY_POLARITY	This is macro CSR_I2S_STREAM_CFG_KEY_POLARITY.
	CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_I2S_STREAM_CFG_KEY_SYNC_RATE.

	CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE.
	CSR_MAX_HQ_PACKET_LEN	This is macro CSR_MAX_HQ_PACKET_LEN.
	CSR_MSG_TYPE_GETREQ	This is macro CSR_MSG_TYPE_GETREQ.
	CSR_MSG_TYPE_GETRESP	This is macro CSR_MSG_TYPE_GETRESP.
	CSR_MSG_TYPE_SETREQ	This is macro CSR_MSG_TYPE_SETREQ.
	CSR_PCM_STREAM_CFG_KEY_LSB_FIRST	This is macro CSR_PCM_STREAM_CFG_KEY_LSB_FIRST.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE.
	CSR_PCM_STREAM_CFG_KEY_MASTER	This is macro CSR_PCM_STREAM_CFG_KEY_MASTER.
	CSR_PCM_STREAM_CFG_KEY_MCLK	This is macro CSR_PCM_STREAM_CFG_KEY_MCLK.
	CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC	This is macro CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC.
	CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND	This is macro CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND.
	CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT	This is macro CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT.
	CSR_PCM_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_PCM_STREAM_CFG_KEY_SYNC_RATE.
	CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE	This is macro CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE.
	CSR_SNK_ADC	This is macro CSR_SNK_ADC.
	CSR_SNK_FASTPIPE	This is macro CSR_SNK_FASTPIPE.
	CSR_SNK_FM	This is macro CSR_SNK_FM.
	CSR_SNK_I2S	This is macro CSR_SNK_I2S.
	CSR_SNK_L2CAP	This is macro CSR_SNK_L2CAP.
	CSR_SNK_PCM	This is macro CSR_SNK_PCM.
	CSR_SNK_SCO	This is macro CSR_SNK_SCO.
	CSR_SNK_SPDIF	This is macro CSR_SNK_SPDIF.
	CSR_SRC_ADC	This is macro CSR_SRC_ADC.
	CSR_SRC_FASTPIPE	This is macro CSR_SRC_FASTPIPE.
	CSR_SRC_FM	This is macro CSR_SRC_FM.
	CSR_SRC_I2S	This is macro CSR_SRC_I2S.
	CSR_SRC_L2CAP	This is macro CSR_SRC_L2CAP.
	CSR_SRC_MIC	This is macro CSR_SRC_MIC.
	CSR_SRC_PCM	This is macro CSR_SRC_PCM.
	CSR_SRC_SCO	This is macro CSR_SRC_SCO.
	CSR_SRC_SPDIF	This is macro CSR_SRC_SPDIF.
	CSR_VARID_CACHED_TEMPERATURE	This is macro CSR_VARID_CACHED_TEMPERATURE.
	CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION	This is macro CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION.
	CSR_VARID_CREATE_OPERATOR_C	This is macro CSR_VARID_CREATE_OPERATOR_C.
	CSR_VARID_CREATE_OPERATOR_P	This is macro CSR_VARID_CREATE_OPERATOR_P.
	CSR_VARID_DESTROY_OPERATORS	This is macro CSR_VARID_DESTROY_OPERATORS.
	CSR_VARID_DSPMANAGER_CONFIG_REQUEST	This is macro CSR_VARID_DSPMANAGER_CONFIG_REQUEST.
	CSR_VARID_ENABLE_SCO_STREAMS	This is macro CSR_VARID_ENABLE_SCO_STREAMS.
	CSR_VARID_MAP_SCO_AUDIO	This is macro CSR_VARID_MAP_SCO_AUDIO.
	CSR_VARID_MESSAGE_FROM_OPERATOR	Select Variable definitions
	CSR_VARID_PIO	This is macro CSR_VARID_PIO.
	CSR_VARID_PIO_DIRECTION_MASK	This is macro CSR_VARID_PIO_DIRECTION_MASK.
	CSR_VARID_PIO_PROTECT_MASK	This is macro CSR_VARID_PIO_PROTECT_MASK.
	CSR_VARID_RSSI_ACL	This is macro CSR_VARID_RSSI_ACL.
	CSR_VARID_START_OPERATORS	This is macro CSR_VARID_START_OPERATORS.
	CSR_VARID_STOP_OPERATORS	This is macro CSR_VARID_STOP_OPERATORS.
	CSR_VARID_STREAM_CLOSE_SINK	This is macro CSR_VARID_STREAM_CLOSE_SINK.
	CSR_VARID_STREAM_CLOSE_SOURCE	This is macro CSR_VARID_STREAM_CLOSE_SOURCE.
	CSR_VARID_STREAM_CONFIGURE	This is macro CSR_VARID_STREAM_CONFIGURE.
	CSR_VARID_STREAM_CONNECT	This is macro CSR_VARID_STREAM_CONNECT.
	CSR_VARID_STREAM_DRAINED_NOTIFICATION	This is macro CSR_VARID_STREAM_DRAINED_NOTIFICATION.
	CSR_VARID_STREAM_GET_SINK	This is macro CSR_VARID_STREAM_GET_SINK.

	CSR_VARID_STREAM_GET_SOURCE	This is macro CSR_VARID_STREAM_GET_SOURCE.
	CSR_VARID_STREAM_TRANSFORM_DISCONNECT	This is macro CSR_VARID_STREAM_TRANSFORM_DISCONNECT.
	GETRESP_BAD_REQ	This is macro GETRESP_BAD_REQ.
	GETRESP_ERROR	This is macro GETRESP_ERROR.
	GETRESP_NO_ACCESS	This is macro GETRESP_NO_ACCESS.
	GETRESP_NO_SUCH_VARID	This is macro GETRESP_NO_SUCH_VARID.
	GETRESP_NO_VALUE	This is macro GETRESP_NO_VALUE.
	GETRESP_OK	BCCMD GETRESP status codes
	GETRESP_PERMISSION_DENIED	This is macro GETRESP_PERMISSION_DENIED.
	GETRESP_READ_ONLY	This is macro GETRESP_READ_ONLY.
	GETRESP_TOO_BIG	This is macro GETRESP_TOO_BIG.
	GETRESP_WRITE_ONLY	This is macro GETRESP_WRITE_ONLY.
	PS_DEFAULT	This is macro PS_DEFAULT.
	PS_F	This is macro PS_F.
	PS_I	This is macro PS_I.
	PS_RAM	This is macro PS_RAM.
	PS_ROM	This is macro PS_ROM.
	PSKEY_ANA_FREQ	This is macro PSKEY_ANA_FREQ.
	PSKEY_BDADDR	Select PS key definitions
	PSKEY_BLE_DEFAULT_TX_POWER	This is macro PSKEY_BLE_DEFAULT_TX_POWER.
	PSKEY_CLOCK_REQUEST_ENABLE	This is macro PSKEY_CLOCK_REQUEST_ENABLE.
	PSKEY_DEEP_SLEEP_CLEAR_RTS	This is macro PSKEY_DEEP_SLEEP_CLEAR_RTS.
	PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO	This is macro PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO.
	PSKEY_DEEP_SLEEP_STATE	This is macro PSKEY_DEEP_SLEEP_STATE.
	PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK	This is macro PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK.
	PSKEY_DEEP_SLEEP_WAKE_CTS	This is macro PSKEY_DEEP_SLEEP_WAKE_CTS.
	PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE	This is macro PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE.
	PSKEY_DIGITAL_AUDIO_CONFIG	This is macro PSKEY_DIGITAL_AUDIO_CONFIG.
	PSKEY_H_HC_FC_MAX_SCO_PKT_LEN	This is macro PSKEY_H_HC_FC_MAX_SCO_PKT_LEN.
	PSKEY_H_HC_FC_MAX_SCO_PKTS	This is macro PSKEY_H_HC_FC_MAX_SCO_PKTS.
	PSKEY_HCI_NOP_DISABLE	This is macro PSKEY_HCI_NOP_DISABLE.
	PSKEY_HOST_INTERFACE	This is macro PSKEY_HOST_INTERFACE.
	PSKEY_HOSTIO_MAP_SCO_PCM	This is macro PSKEY_HOSTIO_MAP_SCO_PCM.
	PSKEY_HOSTIO_UART_RESET_TIMEOUT	This is macro PSKEY_HOSTIO_UART_RESET_TIMEOUT.
	PSKEY_LC_DEFAULT_TX_POWER	This is macro PSKEY_LC_DEFAULT_TX_POWER.
	PSKEY_LC_MAX_TX_POWER	This is macro PSKEY_LC_MAX_TX_POWER.
	PSKEY_LC_MAX_TX_POWER_NO_RSSI	This is macro PSKEY_LC_MAX_TX_POWER_NO_RSSI.
	PSKEY_PCM_CLOCK_RATE	This is macro PSKEY_PCM_CLOCK_RATE.
	PSKEY_PCM_CONFIG32	This is macro PSKEY_PCM_CONFIG32.
	PSKEY_PCM_FORMAT	This is macro PSKEY_PCM_FORMAT.
	PSKEY_PCM_PULL_CONTROL	This is macro PSKEY_PCM_PULL_CONTROL.
	PSKEY_PCM_SYNC_RATE	This is macro PSKEY_PCM_SYNC_RATE.
	PSKEY_PCM_USE_LOW_JITTER_MODE	This is macro PSKEY_PCM_USE_LOW_JITTER_MODE.
	PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL	This is macro PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL.
	PSKEY_UART_BAUDRATE	This is macro PSKEY_UART_BAUDRATE.
	PSKEY_UART_BITRATE	This is macro PSKEY_UART_BITRATE.
	PSKEY_UART_CONFIG_BCSP	This is macro PSKEY_UART_CONFIG_BCSP.
	PSKEY_UART_CONFIG_H4	This is macro PSKEY_UART_CONFIG_H4.
	PSKEY_UART_CONFIG_H4DS	This is macro PSKEY_UART_CONFIG_H4DS.
	PSKEY_UART_CONFIG_H5	This is macro PSKEY_UART_CONFIG_H5.
	PSKEY_UART_TX_WINDOW_SIZE	This is macro PSKEY_UART_TX_WINDOW_SIZE.
	PSKEY_VM_DISABLE	This is macro PSKEY_VM_DISABLE.
	SET_PS_VALUE_BDADDR	This is macro SET_PS_VALUE_BDADDR.

	SET_PS_VALUE_UINT16	Macros for defining lists of PS values for use with btx_csr_set_ps_vars().
	SET_PS_VALUE_UINT32	This is macro SET_PS_VALUE_UINT32.

Structures

	Name	Description
	_btx_csr_autobaud_buffer_t	This is record _btx_csr_autobaud_buffer_t.
	_btx_csr_bccmd_header_s	This is type btx_csr_bccmd_header_t.
	_btx_csr_bccmd_listener_t	This is record _btx_csr_bccmd_listener_t.
	_btx_csr_cached_temperature_s	This is type btx_csr_cached_temperature_t.
	_btx_csr_create_operator_c_s	This is type btx_csr_create_operator_c_t.
	_btx_csr_exec_hq_script_buffer_t	This is record _btx_csr_exec_hq_script_buffer_t.
	_btx_csr_exec_script_buffer_t	This is record _btx_csr_exec_script_buffer_t.
	_btx_csr_pio_direction_mask_s	This is type btx_csr_pio_direction_mask_t.
	_btx_csr_pio_protection_mask_s	This is type btx_csr_pio_protection_mask_t.
	_btx_csr_pio_s	This is type btx_csr_pio_t.
	_btx_csr_rssi_acl_s	This is type btx_csr_rssi_acl_t.
	_btx_csr_script_t	This is type btx_csr_script_t.
	_btx_csr_set_ps_vars_buffer_t	This is record _btx_csr_set_ps_vars_buffer_t.
	_btx_csr_strm_connect_s	This is type btx_csr_strm_connect_t.
	_btx_csr_strm_get_sink_s	This is type btx_csr_strm_get_sink_t.
	_btx_csr_strm_get_source_s	This is type btx_csr_strm_get_source_t.
	btx_csr_bccmd_header_t	This is type btx_csr_bccmd_header_t.
	btx_csr_cached_temperature_t	This is type btx_csr_cached_temperature_t.
	btx_csr_create_operator_c_t	This is type btx_csr_create_operator_c_t.
	btx_csr_pio_direction_mask_t	This is type btx_csr_pio_direction_mask_t.
	btx_csr_pio_protection_mask_t	This is type btx_csr_pio_protection_mask_t.
	btx_csr_pio_t	This is type btx_csr_pio_t.
	btx_csr_rssi_acl_t	This is type btx_csr_rssi_acl_t.
	btx_csr_script_t	This is type btx_csr_script_t.
	btx_csr_strm_connect_t	This is type btx_csr_strm_connect_t.
	btx_csr_strm_get_sink_t	This is type btx_csr_strm_get_sink_t.
	btx_csr_strm_get_source_t	This is type btx_csr_strm_get_source_t.

Types

	Name	Description
	btx_csr_autobaud_buffer_t	This is type btx_csr_autobaud_buffer_t.
	btx_csr_autobaud_callback_fp	This is type btx_csr_autobaud_callback_fp.
	btx_csr_bccmd_callback_fp	This is type btx_csr_bccmd_callback_fp.
	btx_csr_bccmd_listener_t	This is type btx_csr_bccmd_listener_t.
	btx_csr_exec_hq_script_buffer_t	This is type btx_csr_exec_hq_script_buffer_t.
	btx_csr_exec_hq_script_callback_fp	This is type btx_csr_exec_hq_script_callback_fp.
	btx_csr_exec_script_buffer_t	This is type btx_csr_exec_script_buffer_t.
	btx_csr_exec_script_callback_fp	This is type btx_csr_exec_script_callback_fp.
	btx_csr_get_ps_var_callback_fp	This is type btx_csr_get_ps_var_callback_fp.
	btx_csr_get_script_fp	This is type btx_csr_get_script_fp.
	btx_csr_get_var_callback_fp	This is type btx_csr_get_var_callback_fp.
	btx_csr_set_ps_vars_buffer_t	This is type btx_csr_set_ps_vars_buffer_t.
	btx_csr_set_ps_vars_callback_fp	This is type btx_csr_set_ps_vars_callback_fp.
	btx_csr_set_var_callback_fp	This is type btx_csr_set_var_callback_fp.

Unions

	Name	Description
	_btx_csr_var_u	This is type btx_csr_var_t.
	btx_csr_var_t	This is type btx_csr_var_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

frame_buffer.h

Functions

	Name	Description
	bt_l2cap_alloc_frame_buffer	This is function bt_l2cap_alloc_frame_buffer.
	bt_l2cap_free_frame_buffer	This is function bt_l2cap_free_frame_buffer.
	bt_l2cap_init_frame_buffers	This is function bt_l2cap_init_frame_buffers.

Macros

	Name	Description
	L2CAP_MAX_FRAME_BUFFERS	This is macro L2CAP_MAX_FRAME_BUFFERS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

gap.h

Functions

	Name	Description
	bt_gap_find_devices	This is function bt_gap_find_devices.

Structures

	Name	Description
	_bt_device_t	This is type bt_device_t.
	bt_device_t	This is type bt_device_t.

Types

	Name	Description
	bt_find_devices_callback_fp	This is type bt_find_devices_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

hci.h

Functions

	Name	Description
	bt_hci_add_param_bdaddr	brief Add BD address parameter to an HCI command ingroup hci
	bt_hci_add_param_byte	brief Add byte parameter to an HCI command ingroup hci
	bt_hci_add_param_cod	brief Add class of device parameter to an HCI command ingroup hci
	bt_hci_add_param_int	brief Add int parameter to an HCI command ingroup hci
	bt_hci_add_param_linkkey	brief Add link key parameter to an HCI command ingroup hci
	bt_hci_add_param_long	brief Add long parameter to an HCI command ingroup hci
	bt_hci_add_param_string	brief Add string parameter to an HCI command ingroup hci
	bt_hci_alloc_canned_command	brief Allocate and initialize an HCI command structure for a canned (pre-formatted) command. ingroup hci

	bt_hci_alloc_command	brief Allocate and initialize an HCI command structure. ingroup hci
	bt_hci_authenticate_ex	This is function bt_hci_authenticate_ex.
	bt_hci_cancel_find_devices	brief Stop inquiry
	bt_hci_cancel_find_devices_ex	brief Stop inquiry
	bt_hci_cancel_request_remote_name	brief Cancel remote device name request
	bt_hci_cancel_send_acl_data	brief Cancel sending data over ACL connection
	bt_hci_connect	brief Connect to a remote device. ingroup hci details This function tries to establish an HCI connection with a remote device specified by the Bluetooth address c dest. Upon completion, the callback function specified by the c callback parameter is called. param dest Bluetooth address of the remote device. param packet_type param role_switch param acl_config param callback Pointer to a callback function that is called when the connect operation completes. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function... more
	bt_hci_connect_sco	This is function bt_hci_connect_sco.
	bt_hci_disconnect	brief Abort connection ingroup hci
	bt_hci_exit_park_state	brief Exit park state
	bt_hci_exit_sniff_mode_ex	This is function bt_hci_exit_sniff_mode_ex.
	bt_hci_find_devices	brief Start inquiry
	bt_hci_find_devices_ex	This is function bt_hci_find_devices_ex.
	bt_hci_free_command	brief Free HCI command. ingroup hci
	bt_hci_get_evt_param_bdaddr	brief Get bd address parameter from HCI event ingroup hci
	bt_hci_get_evt_param_byte	brief Get byte parameter from HCI event ingroup hci
	bt_hci_get_evt_param_devclass	brief Get class of device parameter from HCI event ingroup hci
	bt_hci_get_evt_param_int	brief Get int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_linkkey	brief Get link key parameter from HCI event ingroup hci
	bt_hci_get_evt_param_long	brief Get long parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	brief Get unsigned int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_ulong	brief Get unsigned long parameter from HCI event ingroup hci
	bt_hci_get_inquiry_response_tx_power_level	This is function bt_hci_get_inquiry_response_tx_power_level.
	bt_hci_get_last_cmd_status	This is function bt_hci_get_last_cmd_status.
	bt_hci_get_param_bdaddr	brief Get BD address parameter from HCI command ingroup hci
	bt_hci_get_param_byte	brief Get byte parameter from HCI command ingroup hci
	bt_hci_get_param_int	brief Get int parameter from HCI command ingroup hci
	bt_hci_get_param_linkkey	brief Get link key parameter from HCI command ingroup hci
	bt_hci_get_param_long	brief Get long parameter from HCI command ingroup hci
	bt_hci_get_recv_buffer	This is function bt_hci_get_recv_buffer.
	bt_hci_get_recv_buffer_len	This is function bt_hci_get_recv_buffer_len.
	bt_hci_get_send_buffer	This is function bt_hci_get_send_buffer.
	bt_hci_get_send_buffer_len	This is function bt_hci_get_send_buffer_len.
	bt_hci_init	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init. This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.
	bt_hci_init_ex	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

	bt_hci_listen	brief Listen for incoming connections. ingroup hci details param callback Pointer to a callback function that is called when a new incoming connection has been established. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_hci_listen_sco	This is function bt_hci_listen_sco.
	bt_hci_park_state	brief Put local device to park state
	bt_hci_read_inquiry_mode	brief Get current inquiry mode
	bt_hci_read_inquiry_scan_activity	brief Get current inquiry scan activity configuration
	bt_hci_read_inquiry_scan_type	brief Get current inquiry scan type
	bt_hci_read_page_scan_activity	brief Get current page scan activity configuration
	bt_hci_read_page_scan_period_mode	brief Get current page scan period mode
	bt_hci_read_page_scan_type	brief Get current page scan type
	bt_hci_read_page_timeout	brief Get current page timeout
	bt_hci_reject_pin_code	This is function bt_hci_reject_pin_code.
	bt_hci_request_remote_name	brief Request remote device's name
	bt_hci_reset	brief Reset controller. ingroup hci details This function resets the BT controller. param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_role_change_ex	brief Change local device's role
	bt_hci_send_acl_data	brief Send data over ACL connection
	bt_hci_send_cmd	brief Send a command to local device
	bt_hci_send_linkkey	This is function bt_hci_send_linkkey.
	bt_hci_send_pin_code_ex	This is function bt_hci_send_pin_code_ex.
	bt_hci_send_sco_data	brief Send data over SCO connection
	bt_hci_set_encryption_ex	brief Set connection encryption
	bt_hci_set_incoming_connection_role	This is function bt_hci_set_incoming_connection_role.
	bt_hci_set_scan	This is function bt_hci_set_scan.
	bt_hci_set_scan_ex	This is function bt_hci_set_scan_ex.
	bt_hci_sniff_mode_ex	This is function bt_hci_sniff_mode_ex.
	bt_hci_sniff_subrating_ex	This is function bt_hci_sniff_subrating_ex.
	bt_hci_start	brief Start HCI layer. ingroup hci details This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps: 1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv. 2. Reset and configure the controller. Upon completion of controller initialization the callback function passed in the c callback parameter is called. param callback Completion callback. Called when controller initialization is complete. param callback_param A pointer to arbitrary data to be passed to the c callback callback. param enable_scan This is a bitmask that defines which scans are enabled... more
	bt_hci_start_no_init	brief Start HCI layer without controller configuration. ingroup hci details This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would... more
	bt_hci_stop	brief Stop HCI layer. ingroup hci details This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack. param callback Completion callback. Called when the HCI layer has been stopped. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

	bt_hci_write_default_link_policy_settings	This is function bt_hci_write_default_link_policy_settings.
	bt_hci_write_inquiry_mode	brief Configure inquiry mode
	bt_hci_write_inquiry_scan_activity	brief Configure inquiry scan activity
	bt_hci_write_inquiry_scan_type	brief Configure inquiry scan type
	bt_hci_write_link_policy_settings	This is function bt_hci_write_link_policy_settings.
	bt_hci_write_local_name	brief Write local device name ingroup hci details The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.
	bt_hci_write_local_name_ex	This is function bt_hci_write_local_name_ex.
	bt_hci_write_page_scan_activity	brief Configure page scan activity
	bt_hci_write_page_scan_period_mode	brief Configure page scan period mode
	bt_hci_write_page_scan_type	brief Configure page scan type
	bt_hci_write_page_timeout	brief Configure page timeout
	hci_sleep	brief Put local device to sleep
	hci_wakeup	brief Awaken local device

Macros

	Name	Description
	bt_hci_add_param_hconn	brief Add connection handle parameter to an HCI command ingroup hci
	bt_hci_add_param_lap	This is macro bt_hci_add_param_lap.
	bt_hci_add_param_uint	brief Add unsigned int parameter to an HCI command ingroup hci
	bt_hci_add_param_ulong	brief Add unsigned long parameter to an HCI command ingroup hci
	bt_hci_authenticate	This is macro bt_hci_authenticate.
	bt_hci_exit_sniff_mode	brief Cancel sniff mode
	bt_hci_get_evt_param_hconn	brief Get connection handle parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	This is macro bt_hci_get_evt_param_uint.
	bt_hci_get_evt_param_ulong	This is macro bt_hci_get_evt_param_ulong.
	bt_hci_get_param_hconn	brief Get connection handle parameter from HCI command ingroup hci
	bt_hci_role_change	This is macro bt_hci_role_change.
	bt_hci_send_pin_code	This is macro bt_hci_send_pin_code.
	bt_hci_set_encryption	This is macro bt_hci_set_encryption.
	bt_hci_sniff_mode	brief Put local device to sniff mode
	bt_hci_sniff_subrating	This is macro bt_hci_sniff_subrating.
	HCI_ACCEPT_CONNECTION_REQUEST	This is macro HCI_ACCEPT_CONNECTION_REQUEST.
	HCI_ACCEPT_SYNCH_CONNECTION_REQUEST	This is macro HCI_ACCEPT_SYNCH_CONNECTION_REQUEST.
	HCI_ACL_DATA_HEADER_LEN	This is macro HCI_ACL_DATA_HEADER_LEN.
	HCI_AUTHENTICATION_REQUESTED	This is macro HCI_AUTHENTICATION_REQUESTED.
	HCI_BB_PACKET_TYPE_DH1	This is macro HCI_BB_PACKET_TYPE_DH1.
	HCI_BB_PACKET_TYPE_DH3	This is macro HCI_BB_PACKET_TYPE_DH3.
	HCI_BB_PACKET_TYPE_DH5	This is macro HCI_BB_PACKET_TYPE_DH5.
	HCI_BB_PACKET_TYPE_DM1	This is macro HCI_BB_PACKET_TYPE_DM1.
	HCI_BB_PACKET_TYPE_DM3	This is macro HCI_BB_PACKET_TYPE_DM3.
	HCI_BB_PACKET_TYPE_DM5	This is macro HCI_BB_PACKET_TYPE_DM5.
	HCI_BB_PACKET_TYPE_NO_2_DH1	This is macro HCI_BB_PACKET_TYPE_NO_2_DH1.
	HCI_BB_PACKET_TYPE_NO_2_DH3	This is macro HCI_BB_PACKET_TYPE_NO_2_DH3.
	HCI_BB_PACKET_TYPE_NO_2_DH5	This is macro HCI_BB_PACKET_TYPE_NO_2_DH5.
	HCI_BB_PACKET_TYPE_NO_3_DH1	This is macro HCI_BB_PACKET_TYPE_NO_3_DH1.
	HCI_BB_PACKET_TYPE_NO_3_DH3	This is macro HCI_BB_PACKET_TYPE_NO_3_DH3.
	HCI_BB_PACKET_TYPE_NO_3_DH5	This is macro HCI_BB_PACKET_TYPE_NO_3_DH5.

	HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_P2P	This is macro HCI_C2H_BROADCAST_P2P.
	HCI_C2H_BROADCAST_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_RESERVED	This is macro HCI_C2H_BROADCAST_RESERVED.
	HCI_CHANGE_CONNECTION_LINK_KEY	This is macro HCI_CHANGE_CONNECTION_LINK_KEY.
	HCI_CHANGE_CONNECTION_PACKET_TYPE	This is macro HCI_CHANGE_CONNECTION_PACKET_TYPE.
	HCI_CMD_HEADER_LEN	This is macro HCI_CMD_HEADER_LEN.
	HCI_CMD_STATUS_BEING_SENT	This is macro HCI_CMD_STATUS_BEING_SENT.
	HCI_CMD_STATUS_PENDING	This is macro HCI_CMD_STATUS_PENDING.
	HCI_CMD_STATUS_WAITING_RESPONSE	This is macro HCI_CMD_STATUS_WAITING_RESPONSE.
	HCI_CONFIG_BECOME_MASTER	This is macro HCI_CONFIG_BECOME_MASTER.
	HCI_CONFIG_ENABLE_AUTHENTICATION	This is macro HCI_CONFIG_ENABLE_AUTHENTICATION.
	HCI_CONFIG_ENABLE_ENCRYPTION	This is macro HCI_CONFIG_ENABLE_ENCRYPTION.
	HCI_CONNECTABLE	This is macro HCI_CONNECTABLE.
	HCI_CONTROLLER	This is macro HCI_CONTROLLER.
	HCI_CREATE_CONNECTION	This is macro HCI_CREATE_CONNECTION.
	HCI_CREATE_CONNECTION_CANCEL	This is macro HCI_CREATE_CONNECTION_CANCEL.
	HCI_CREATE_NEW_UNIT_KEY	This is macro HCI_CREATE_NEW_UNIT_KEY.
	HCI_DATA_STATUS_BEING_SENT	This is macro HCI_DATA_STATUS_BEING_SENT.
	HCI_DATA_STATUS_PENDING	This is macro HCI_DATA_STATUS_PENDING.
	HCI_DEFAULT_ACL_CONFIG	Default value for acl_config parameter of bt_hci_connect()
	HCI_DELETE_STORED_LINK_KEY	This is macro HCI_DELETE_STORED_LINK_KEY.
	HCI_DISCONNECT	This is macro HCI_DISCONNECT.
	HCI_DISCOVERABLE	This is macro HCI_DISCOVERABLE.
	HCI_DISCOVERABLE_MODE_GENERAL	This is macro HCI_DISCOVERABLE_MODE_GENERAL.
	HCI_DISCOVERABLE_MODE_LIMITED	This is macro HCI_DISCOVERABLE_MODE_LIMITED.
	HCI_ENABLE_DEVICE_UNDER_TEST_MODE	This is macro HCI_ENABLE_DEVICE_UNDER_TEST_MODE.
	HCI_ENCRYPTION_OFF	This is macro HCI_ENCRYPTION_OFF.
	HCI_ENCRYPTION_ON	This is macro HCI_ENCRYPTION_ON.
	HCI_ENHANCED_FLUSH	This is macro HCI_ENHANCED_FLUSH.
	HCI_EVT_ALL_HCI_EVENTS	Used in bt_hci_ctrl_register_event_listener to indicate that the listener is called for all HCI events (excluding dotstack internal ones)
	HCI_EVT_AUTHENTICATION_COMPLETE	This is macro HCI_EVT_AUTHENTICATION_COMPLETE.
	HCI_EVT_CHANGE_CONN_LINK_COMPLETE	This is macro HCI_EVT_CHANGE_CONN_LINK_COMPLETE.
	HCI_EVT_CMD_SEND_FINISHED	This is macro HCI_EVT_CMD_SEND_FINISHED.
	HCI_EVT_CMD_SEND_STARTED	dotstack internal events
	HCI_EVT_COMMAND_COMPLETE	This is macro HCI_EVT_COMMAND_COMPLETE.
	HCI_EVT_COMMAND_COMPLETE_PARAM_LEN	This is macro HCI_EVT_COMMAND_COMPLETE_PARAM_LEN.
	HCI_EVT_COMMAND_STATUS	This is macro HCI_EVT_COMMAND_STATUS.
	HCI_EVT_CONN_PACKET_TYPE_CHANGED	This is macro HCI_EVT_CONN_PACKET_TYPE_CHANGED.
	HCI_EVT_CONNECTION_COMPLETE	This is macro HCI_EVT_CONNECTION_COMPLETE.
	HCI_EVT_CONNECTION_REQUEST	This is macro HCI_EVT_CONNECTION_REQUEST.
	HCI_EVT_DATA_BUFFER_OVERFLOW	This is macro HCI_EVT_DATA_BUFFER_OVERFLOW.
	HCI_EVT_DISCONNECTION_COMPLETE	This is macro HCI_EVT_DISCONNECTION_COMPLETE.

	HCI_EVT_ENCRYPTION_CHANGE	This is macro HCI_EVT_ENCRYPTION_CHANGE.
	HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE	This is macro HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE.
	HCI_EVT_ENHANCED_FLUSH_COMPLETE	This is macro HCI_EVT_ENHANCED_FLUSH_COMPLETE.
	HCI_EVT_EXTENDED_INQUIRY_RESULT	This is macro HCI_EVT_EXTENDED_INQUIRY_RESULT.
	HCI_EVT_FIRST	This is macro HCI_EVT_FIRST.
	HCI_EVT_FLOW_SPECIFICATION_COMPLETE	This is macro HCI_EVT_FLOW_SPECIFICATION_COMPLETE.
	HCI_EVT_FLUSH_OCCURED	This is macro HCI_EVT_FLUSH_OCCURED.
	HCI_EVT_HARDWARE_ERROR	This is macro HCI_EVT_HARDWARE_ERROR.
	HCI_EVT_INQUIRY_COMPLETE	• addtogroup hci • @{ • @name Events
	HCI_EVT_INQUIRY_RESULT	This is macro HCI_EVT_INQUIRY_RESULT.
	HCI_EVT_INQUIRY_RESULT_WITH_RSSI	This is macro HCI_EVT_INQUIRY_RESULT_WITH_RSSI.
	HCI_EVT_IO_CAPABILITY_REQUEST	This is macro HCI_EVT_IO_CAPABILITY_REQUEST.
	HCI_EVT_IO_CAPABILITY_RESPONSE	This is macro HCI_EVT_IO_CAPABILITY_RESPONSE.
	HCI_EVT_KEYPRESS_NOTIFICATION	This is macro HCI_EVT_KEYPRESS_NOTIFICATION.
	HCI_EVT_LAST	This is macro HCI_EVT_LAST.
	HCI_EVT_LE_META_EVENT	This is macro HCI_EVT_LE_META_EVENT.
	HCI_EVT_LINK_IS_BUSY	This is macro HCI_EVT_LINK_IS_BUSY.
	HCI_EVT_LINK_IS_IDLE	This is macro HCI_EVT_LINK_IS_IDLE.
	HCI_EVT_LINK_KEY_NOTIFICATION	This is macro HCI_EVT_LINK_KEY_NOTIFICATION.
	HCI_EVT_LINK_KEY_REQUEST	This is macro HCI_EVT_LINK_KEY_REQUEST.
	HCI_EVT_LINK_SUPERVISION_TO_CHANGED	This is macro HCI_EVT_LINK_SUPERVISION_TO_CHANGED.
	HCI_EVT_LOOPBACK_COMMAND	This is macro HCI_EVT_LOOPBACK_COMMAND.
	HCI_EVT_MASTER_LINK_KEY_COMPLETE	This is macro HCI_EVT_MASTER_LINK_KEY_COMPLETE.
	HCI_EVT_MAX_SLOTS_CHANGE	This is macro HCI_EVT_MAX_SLOTS_CHANGE.
	HCI_EVT_MODE_CHANGE	This is macro HCI_EVT_MODE_CHANGE.
	HCI_EVT_NUM_OF_COMPLETED_PACKETS	This is macro HCI_EVT_NUM_OF_COMPLETED_PACKETS.
	HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE	This is macro HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE.
	HCI_EVT_PIN_CODE_REQUEST	This is macro HCI_EVT_PIN_CODE_REQUEST.
	HCI_EVT_QOS_SETUP_COMPLETE	This is macro HCI_EVT_QOS_SETUP_COMPLETE.
	HCI_EVT_QOS_VIOLATION	This is macro HCI_EVT_QOS_VIOLATION.
	HCI_EVT_READ_CLOCK_OFFSET_COMPLETE	This is macro HCI_EVT_READ_CLOCK_OFFSET_COMPLETE.
	HCI_EVT_READ_RMT_EXT_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_EXT_FEATURES_COMP.
	HCI_EVT_READ_RMT_SUP_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_SUP_FEATURES_COMP.
	HCI_EVT_READ_RMT_VERSION_INFO_COMP	This is macro HCI_EVT_READ_RMT_VERSION_INFO_COMP.
	HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE	This is macro HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE.
	HCI_EVT_REMOTE_OOB_DATA_REQUEST	This is macro HCI_EVT_REMOTE_OOB_DATA_REQUEST.
	HCI_EVT_RETURN_LINK_KEYS	This is macro HCI_EVT_RETURN_LINK_KEYS.
	HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF	This is macro HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF.
	HCI_EVT_ROLE_CHANGE	This is macro HCI_EVT_ROLE_CHANGE.
	HCI_EVT_SIMPLE_PAIRING_COMPLETE	This is macro HCI_EVT_SIMPLE_PAIRING_COMPLETE.

	HCI_EVT_SNIFF_SUBRATING	This is macro HCI_EVT_SNIFF_SUBRATING.
	HCI_EVT_SYNCH_CONNECTION_CHANGED	This is macro HCI_EVT_SYNCH_CONNECTION_CHANGED.
	HCI_EVT_SYNCH_CONNECTION_COMPLETE	This is macro HCI_EVT_SYNCH_CONNECTION_COMPLETE.
	HCI_EVT_USER_CONFIRMATION_REQUEST	This is macro HCI_EVT_USER_CONFIRMATION_REQUEST.
	HCI_EVT_USER_PASSKEY_NOTIFICATION	This is macro HCI_EVT_USER_PASSKEY_NOTIFICATION.
	HCI_EVT_USER_PASSKEY_REQUEST	This is macro HCI_EVT_USER_PASSKEY_REQUEST.
	HCI_EXIT_PARK_STATE	This is macro HCI_EXIT_PARK_STATE.
	HCI_EXIT_PERIODIC_INQUIRY_MODE	This is macro HCI_EXIT_PERIODIC_INQUIRY_MODE.
	HCI_EXIT_SNIFF_MODE	This is macro HCI_EXIT_SNIFF_MODE.
	HCI_FLOW_SPECIFICATION	This is macro HCI_FLOW_SPECIFICATION.
	HCI_FLUSH	This is macro HCI_FLUSH.
	HCI_H2C_BROADCAST_ACTIVE_SLAVE	This is macro HCI_H2C_BROADCAST_ACTIVE_SLAVE.
	HCI_H2C_BROADCAST_NO_BROADCATS	This is macro HCI_H2C_BROADCAST_NO_BROADCATS.
	HCI_H2C_BROADCAST_PARCKED_SLAVE	This is macro HCI_H2C_BROADCAST_PARCKED_SLAVE.
	HCI_H2C_BROADCAST_RESERVED	This is macro HCI_H2C_BROADCAST_RESERVED.
	HCI_HOLD_MODE	addtogroup hci @{ @name Link policy commands details The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.
	HCI_HOST_BUFFER_SIZE	This is macro HCI_HOST_BUFFER_SIZE.
	HCI_HOST_NUM_OF_COMPLETED_PACKETS	This is macro HCI_HOST_NUM_OF_COMPLETED_PACKETS.
	HCI_INQUIRY	addtogroup hci @{ @name Link control commands details The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.
	HCI_INQUIRY_CANCEL	This is macro HCI_INQUIRY_CANCEL.
	HCI_INQUIRY_MODE_EXTENDED	This is macro HCI_INQUIRY_MODE_EXTENDED.
	HCI_INQUIRY_MODE_STANDARD	This is macro HCI_INQUIRY_MODE_STANDARD.
	HCI_INQUIRY_MODE_WITH_RSSI	This is macro HCI_INQUIRY_MODE_WITH_RSSI.
	HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY.
	HCI_IO_CAPABILITY_REQUEST_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_REPLY.
	HCI_LE_ADD_DEVICE_TO_WHITE_LIST	This is macro HCI_LE_ADD_DEVICE_TO_WHITE_LIST.
	HCI_LE_CLEAR_WHITE_LIST	This is macro HCI_LE_CLEAR_WHITE_LIST.
	HCI_LE_CONNECTION_UPDATE	This is macro HCI_LE_CONNECTION_UPDATE.
	HCI_LE_CREATE_CONNECTION	This is macro HCI_LE_CREATE_CONNECTION.
	HCI_LE_CREATE_CONNECTION_CANCEL	This is macro HCI_LE_CREATE_CONNECTION_CANCEL.
	HCI_LE_ENCRYPT	This is macro HCI_LE_ENCRYPT.
	HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LE_LONG_TERM_KEY_REQUEST_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_REPLY.
	HCI_LE_MAX_AD_LEN	This is macro HCI_LE_MAX_AD_LEN.
	HCI_LE_RAND	This is macro HCI_LE_RAND.
	HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER	This is macro HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER.
	HCI_LE_READ_BUFFER_SIZE	This is macro HCI_LE_READ_BUFFER_SIZE.

	HCI_LE_READ_CHANNEL_MAP	This is macro HCI_LE_READ_CHANNEL_MAP.
	HCI_LE_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_LE_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_LE_READ_REMOTE_USED_FEATURES	This is macro HCI_LE_READ_REMOTE_USED_FEATURES.
	HCI_LE_READ_SUPPORTED_STATES	This is macro HCI_LE_READ_SUPPORTED_STATES.
	HCI_LE_READ_WHITE_LIST_SIZE	This is macro HCI_LE_READ_WHITE_LIST_SIZE.
	HCI_LE_RECEIVE_TEST	This is macro HCI_LE_RECEIVE_TEST.
	HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST	This is macro HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST.
	HCI_LE_SET_ADVERTISE_ENABLE	This is macro HCI_LE_SET_ADVERTISE_ENABLE.
	HCI_LE_SET_ADVERTISING_DATA	This is macro HCI_LE_SET_ADVERTISING_DATA.
	HCI_LE_SET_ADVERTISING_PARAMETERS	This is macro HCI_LE_SET_ADVERTISING_PARAMETERS.
	HCI_LE_SET_EVENT_MASK	addtogroup hci @{ @name LE controller commands details The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.
	HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION.
	HCI_LE_SET_RANDOM_ADDRESS	This is macro HCI_LE_SET_RANDOM_ADDRESS.
	HCI_LE_SET_SCAN_ENABLE	This is macro HCI_LE_SET_SCAN_ENABLE.
	HCI_LE_SET_SCAN_PARAMETERS	This is macro HCI_LE_SET_SCAN_PARAMETERS.
	HCI_LE_SET_SCAN_RESPONSE_DATA	This is macro HCI_LE_SET_SCAN_RESPONSE_DATA.
	HCI_LE_START_ENCRYPTION	This is macro HCI_LE_START_ENCRYPTION.
	HCI_LE_TEST_END	This is macro HCI_LE_TEST_END.
	HCI_LE_TRANSMITTER_TEST	This is macro HCI_LE_TRANSMITTER_TEST.
	HCI_LINK_KEY_LEN	This is macro HCI_LINK_KEY_LEN.
	HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LINK_KEY_REQUEST_REPLY	This is macro HCI_LINK_KEY_REQUEST_REPLY.
	HCI_LINK_KEY_TYPE_COMBINATION	This is macro HCI_LINK_KEY_TYPE_COMBINATION.
	HCI_LINK_KEY_TYPE_LOCAL_UNIT	This is macro HCI_LINK_KEY_TYPE_LOCAL_UNIT.
	HCI_LINK_KEY_TYPE_REMOTE_UNIT	This is macro HCI_LINK_KEY_TYPE_REMOTE_UNIT.
	HCI_LINK_POLICY_ENABLE_ALL	This is macro HCI_LINK_POLICY_ENABLE_ALL.
	HCI_LINK_POLICY_ENABLE_HOLD_MODE	This is macro HCI_LINK_POLICY_ENABLE_HOLD_MODE.
	HCI_LINK_POLICY_ENABLE_PARK_STATE	This is macro HCI_LINK_POLICY_ENABLE_PARK_STATE.
	HCI_LINK_POLICY_ENABLE_ROLE_SWITCH	This is macro HCI_LINK_POLICY_ENABLE_ROLE_SWITCH.
	HCI_LINK_POLICY_ENABLE_SNIFF_MODE	This is macro HCI_LINK_POLICY_ENABLE_SNIFF_MODE.
	HCI_MASTER_LINK_KEY	This is macro HCI_MASTER_LINK_KEY.
	HCI_MAX_EVENT_PARAM_LEN	This is macro HCI_MAX_EVENT_PARAM_LEN.
	HCI_MAX_PARAM_LEN	This is macro HCI_MAX_PARAM_LEN.
	HCI_MAX_PIN_LENGTH	This is macro HCI_MAX_PIN_LENGTH.
	HCI_OPCODE	This is macro HCI_OPCODE.
	HCI_PACKET_BOUNDARY_COMPLETE	This is macro HCI_PACKET_BOUNDARY_COMPLETE.
	HCI_PACKET_BOUNDARY_CONTINUE	This is macro HCI_PACKET_BOUNDARY_CONTINUE.
	HCI_PACKET_BOUNDARY_FIRST	This is macro HCI_PACKET_BOUNDARY_FIRST.
	HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH.
	HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH .
	HCI_PACKET_TYPE_ACL_DATA	This is macro HCI_PACKET_TYPE_ACL_DATA.

	HCI_PACKET_TYPE_COMMAND	This is macro HCI_PACKET_TYPE_COMMAND.
	HCI_PACKET_TYPE_EVENT	This is macro HCI_PACKET_TYPE_EVENT.
	HCI_PACKET_TYPE_NONE	This is macro HCI_PACKET_TYPE_NONE.
	HCI_PACKET_TYPE_SCO_DATA	This is macro HCI_PACKET_TYPE_SCO_DATA.
	HCI_PAGE_SCAN_REPETITION_MODE_R0	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R0.
	HCI_PAGE_SCAN_REPETITION_MODE_R1	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R1.
	HCI_PAGE_SCAN_REPETITION_MODE_R2	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R2.
	HCI_PARAM_LEN_BD_ADDR	This is macro HCI_PARAM_LEN_BD_ADDR.
	HCI_PARAM_LEN_BYTE	This is macro HCI_PARAM_LEN_BYTE.
	HCI_PARAM_LEN_DEV_CLASS	This is macro HCI_PARAM_LEN_DEV_CLASS.
	HCI_PARAM_LEN_HANDLE	This is macro HCI_PARAM_LEN_HANDLE.
	HCI_PARAM_LEN_INT	This is macro HCI_PARAM_LEN_INT.
	HCI_PARAM_LEN_LONG	This is macro HCI_PARAM_LEN_LONG.
	HCI_PARAM_TYPE_BD_ADDR	This is macro HCI_PARAM_TYPE_BD_ADDR.
	HCI_PARAM_TYPE_BYTE	This is macro HCI_PARAM_TYPE_BYTE.
	HCI_PARAM_TYPE_DEV_CLASS	This is macro HCI_PARAM_TYPE_DEV_CLASS.
	HCI_PARAM_TYPE_HANDLE	This is macro HCI_PARAM_TYPE_HANDLE.
	HCI_PARAM_TYPE_INT	This is macro HCI_PARAM_TYPE_INT.
	HCI_PARAM_TYPE_LONG	This is macro HCI_PARAM_TYPE_LONG.
	HCI_PARAM_TYPE_STRING	This is macro HCI_PARAM_TYPE_STRING.
	HCI_PARK_STATE	This is macro HCI_PARK_STATE.
	HCI_PERIODIC_INQUIRY_MODE	This is macro HCI_PERIODIC_INQUIRY_MODE.
	HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY	This is macro HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY.
	HCI_PIN_CODE_REQUEST_REPLY	This is macro HCI_PIN_CODE_REQUEST_REPLY.
	HCI_POWER_MODE_ACTIVE	This is macro HCI_POWER_MODE_ACTIVE.
	HCI_POWER_MODE_HOLD	This is macro HCI_POWER_MODE_HOLD.
	HCI_POWER_MODE_PARK	This is macro HCI_POWER_MODE_PARK.
	HCI_POWER_MODE_SNIFF	This is macro HCI_POWER_MODE_SNIFF.
	HCI_QOS_SETUP	This is macro HCI_QOS_SETUP.
	HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_READ_AFH_CHANNEL_MAP	This is macro HCI_READ_AFH_CHANNEL_MAP.
	HCI_READ_AUTHENTICATION_ENABLE	This is macro HCI_READ_AUTHENTICATION_ENABLE.
	HCI_READ_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_READ_AUTOMATIC_FLASH_TIMEOUT.
	HCI_READ_BD_ADDR	This is macro HCI_READ_BD_ADDR.
	HCI_READ_BUFFER_SIZE	This is macro HCI_READ_BUFFER_SIZE.
	HCI_READ_CLASS_OF_DEVICE	This is macro HCI_READ_CLASS_OF_DEVICE.
	HCI_READ_CLOCK_COMMAND	This is macro HCI_READ_CLOCK_COMMAND.
	HCI_READ_CLOCK_OFFSET	This is macro HCI_READ_CLOCK_OFFSET.
	HCI_READ_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_READ_CONNECTION_ACCEPT_TIMEOUT.
	HCI_READ_CURRENT_IAC_LAP	This is macro HCI_READ_CURRENT_IAC_LAP.
	HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_READ_DEFAULT_POLICY_SETTINGS	This is macro HCI_READ_DEFAULT_POLICY_SETTINGS.
	HCI_READ_ENCRYPTION_MODE	This is macro HCI_READ_ENCRYPTION_MODE.
	HCI_READ_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_READ_EXTENDED_INQUIRY_RESPONSE.

	HCI_READ_FAILED_CONTACT_COUNTER	addtogroup hci @{ @name Status Parameters details The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.
	HCI_READ_HOLD_MODE_ACTIVITY	This is macro HCI_READ_HOLD_MODE_ACTIVITY.
	HCI_READ_INQUIRY_MODE	This is macro HCI_READ_INQUIRY_MODE.
	HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL	This is macro HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL.
	HCI_READ_INQUIRY_SCAN_ACTIVITY	This is macro HCI_READ_INQUIRY_SCAN_ACTIVITY.
	HCI_READ_INQUIRY_SCAN_TYPE	This is macro HCI_READ_INQUIRY_SCAN_TYPE.
	HCI_READ_LE_HOST_SUPPORT	This is macro HCI_READ_LE_HOST_SUPPORT.
	HCI_READ_LINK_POLICY_SETTINGS	This is macro HCI_READ_LINK_POLICY_SETTINGS.
	HCI_READ_LINK_QUALITY	This is macro HCI_READ_LINK_QUALITY.
	HCI_READ_LINK_SUPERVISION_TIMEOUT	This is macro HCI_READ_LINK_SUPERVISION_TIMEOUT.
	HCI_READ_LMP_HANDLE	This is macro HCI_READ_LMP_HANDLE.
	HCI_READ_LOCAL_EXTENDED_FEATURES	This is macro HCI_READ_LOCAL_EXTENDED_FEATURES.
	HCI_READ_LOCAL_NAME	This is macro HCI_READ_LOCAL_NAME.
	HCI_READ_LOCAL_OOB_DATA	This is macro HCI_READ_LOCAL_OOB_DATA.
	HCI_READ_LOCAL_SUPPORTED_COMMANDS	This is macro HCI_READ_LOCAL_SUPPORTED_COMMANDS.
	HCI_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_READ_LOCAL_VERSION_INFORMATION	addtogroup hci @{ @name Informational Parameters details The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.
	HCI_READ_LOOPBACK_MODE	addtogroup hci @{ @name Testing Commands details The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.
	HCI_READ_NUM_BROADCST_RETR	This is macro HCI_READ_NUM_BROADCST_RETR.
	HCI_READ_NUM_OF_SUPPORTED_IAC	This is macro HCI_READ_NUM_OF_SUPPORTED_IAC.
	HCI_READ_PAGE_SCAN_ACTIVITY	This is macro HCI_READ_PAGE_SCAN_ACTIVITY.
	HCI_READ_PAGE_SCAN_PERIOD_MODE	This is macro HCI_READ_PAGE_SCAN_PERIOD_MODE.
	HCI_READ_PAGE_SCAN_TYPE	This is macro HCI_READ_PAGE_SCAN_TYPE.
	HCI_READ_PAGE_TIMEOUT	This is macro HCI_READ_PAGE_TIMEOUT.
	HCI_READ_PIN_TYPE	This is macro HCI_READ_PIN_TYPE.
	HCI_READ_REFRESH_ENCRYPTION_KEY	This is macro HCI_READ_REFRESH_ENCRYPTION_KEY.
	HCI_READ_REMOTE_EXTENDED_FEATURES	This is macro HCI_READ_REMOTE_EXTENDED_FEATURES.
	HCI_READ_REMOTE_SUPPORTED_FEATURES	This is macro HCI_READ_REMOTE_SUPPORTED_FEATURES.
	HCI_READ_REMOTE_VERSION_INFORMATION	This is macro HCI_READ_REMOTE_VERSION_INFORMATION.
	HCI_READ_RSSI	This is macro HCI_READ_RSSI.
	HCI_READ_SCAN_ENABLE	This is macro HCI_READ_SCAN_ENABLE.

	HCI_READ_SIMPLE_PAIRING_MODE	This is macro HCI_READ_SIMPLE_PAIRING_MODE.
	HCI_READ_STORED_LINK_KEY	This is macro HCI_READ_STORED_LINK_KEY.
	HCI_READ_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_READ_SYNC_FLOW_CONTROL_ENABLE.
	HCI_READ_TRANSMIT_POWER_LEVEL	This is macro HCI_READ_TRANSMIT_POWER_LEVEL.
	HCI_READ_VOICE_SETTING	This is macro HCI_READ_VOICE_SETTING.
	HCI_REJECT_CONNECTION_REQUEST	This is macro HCI_REJECT_CONNECTION_REQUEST.
	HCI_REJECT_SYNCH_CONNECTION_REQUEST	This is macro HCI_REJECT_SYNCH_CONNECTION_REQUEST.
	HCI_REMOTE_NAME_REQUEST	This is macro HCI_REMOTE_NAME_REQUEST.
	HCI_REMOTE_NAME_REQUEST_CANCEL	This is macro HCI_REMOTE_NAME_REQUEST_CANCEL.
	HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY.
	HCI_REMOTE_OOB_DATA_REQUEST_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_REPLY.
	HCI_RESET	This is macro HCI_RESET.
	HCI_RESET_FAILED_CONTACT_COUNTER	This is macro HCI_RESET_FAILED_CONTACT_COUNTER.
	HCI_ROLE_DISCOVERY	This is macro HCI_ROLE_DISCOVERY.
	HCI_ROLE_SWITCH_ALLOW	This is macro HCI_ROLE_SWITCH_ALLOW.
	HCI_ROLE_SWITCH_DISALLOW	This is macro HCI_ROLE_SWITCH_DISALLOW.
	HCI_SCAN_INQUIRY	Flags for HCI_READ_SCAN_ENABLE/HCI_WRITE_SCAN_ENABLE
	HCI_SCAN_PAGE	This is macro HCI_SCAN_PAGE.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED.
	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16.

	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8.
	HCI_SCO_DATA_HEADER_LEN	This is macro HCI_SCO_DATA_HEADER_LEN.
	HCI_SCO_MAX_DATA_LEN	This is macro HCI_SCO_MAX_DATA_LEN.
	HCI_SCO_MAX_LATENCY_DONTCARE	This is macro HCI_SCO_MAX_LATENCY_DONTCARE.
	HCI_SCO_PACKET_TYPE_ALL	This is macro HCI_SCO_PACKET_TYPE_ALL.
	HCI_SCO_PACKET_TYPE_EV3	This is macro HCI_SCO_PACKET_TYPE_EV3.
	HCI_SCO_PACKET_TYPE_EV4	This is macro HCI_SCO_PACKET_TYPE_EV4.
	HCI_SCO_PACKET_TYPE_EV5	This is macro HCI_SCO_PACKET_TYPE_EV5.
	HCI_SCO_PACKET_TYPE_HV1	This is macro HCI_SCO_PACKET_TYPE_HV1.
	HCI_SCO_PACKET_TYPE_HV2	This is macro HCI_SCO_PACKET_TYPE_HV2.
	HCI_SCO_PACKET_TYPE_HV3	This is macro HCI_SCO_PACKET_TYPE_HV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV5.
	HCI_SCO_PACKET_TYPE_NO_3_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV3.
	HCI_SCO_PACKET_TYPE_NO_3_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV5.
	HCI_SCO_RTX_EFFORT_DONTCARE	This is macro HCI_SCO_RTX_EFFORT_DONTCARE.
	HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION	This is macro HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION.
	HCI_SCO_RX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_RX_BANDWIDTH_DONTCARE.
	HCI_SCO_TX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_TX_BANDWIDTH_DONTCARE.
	HCI_SEND_DATA_STATUS_INTERRUPTED	This is macro HCI_SEND_DATA_STATUS_INTERRUPTED.
	HCI_SEND_DATA_STATUS_SUCCESS	This is macro HCI_SEND_DATA_STATUS_SUCCESS.
	HCI_SEND_KEY_PRESS_NOTIFICATION	This is macro HCI_SEND_KEY_PRESS_NOTIFICATION.
	HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION.
	HCI_SET_CONNECTION_ENCRYPTION	This is macro HCI_SET_CONNECTION_ENCRYPTION.
	HCI_SET_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_SET_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_SET_EVENT_FILTER	This is macro HCI_SET_EVENT_FILTER.
	HCI_SET_EVENT_MASK	addtogroup hci @{ @name Controller & Baseband commands details The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.
	HCI_SETUP_SYNCHRONOUS_CONNECTION	This is macro HCI_SETUP_SYNCHRONOUS_CONNECTION.
	HCI_SNIFF_MODE	This is macro HCI_SNIFF_MODE.
	HCI_SNIFF_SUBRATING	This is macro HCI_SNIFF_SUBRATING.
	HCI_SWITCH_ROLE	This is macro HCI_SWITCH_ROLE.
	HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY	This is macro HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY.
	HCI_USER_CONFIRMATION_REQUEST_REPLY	This is macro HCI_USER_CONFIRMATION_REQUEST_REPLY.
	HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY.
	HCI_USER_PASSKEY_REQUEST_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_REPLY.

	HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_WRITE_AUTHENTICATION_ENABLE	This is macro HCI_WRITE_AUTHENTICATION_ENABLE.
	HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT.
	HCI_WRITE_CLASS_OF_DEVICE	This is macro HCI_WRITE_CLASS_OF_DEVICE.
	HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT.
	HCI_WRITE_CURRENT_IAC_LAP	This is macro HCI_WRITE_CURRENT_IAC_LAP.
	HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_WRITE_DEFAULT_POLICY_SETTINGS	This is macro HCI_WRITE_DEFAULT_POLICY_SETTINGS.
	HCI_WRITE_ENCRYPTION_MODE	This is macro HCI_WRITE_ENCRYPTION_MODE.
	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE.
	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN.
	HCI_WRITE_HOLD_MODE_ACTIVITY	This is macro HCI_WRITE_HOLD_MODE_ACTIVITY.
	HCI_WRITE_INQUIRY_MODE	This is macro HCI_WRITE_INQUIRY_MODE.
	HCI_WRITE_INQUIRY_SCAN_ACTIVITY	This is macro HCI_WRITE_INQUIRY_SCAN_ACTIVITY.
	HCI_WRITE_INQUIRY_SCAN_TYPE	This is macro HCI_WRITE_INQUIRY_SCAN_TYPE.
	HCI_WRITE_INQUIRY_TX_POWER_LEVEL	This is macro HCI_WRITE_INQUIRY_TX_POWER_LEVEL.
	HCI_WRITE_LE_HOST_SUPPORT	This is macro HCI_WRITE_LE_HOST_SUPPORT.
	HCI_WRITE_LINK_POLICY_SETTINGS	This is macro HCI_WRITE_LINK_POLICY_SETTINGS.
	HCI_WRITE_LINK_SUPERVISION_TIMEOUT	This is macro HCI_WRITE_LINK_SUPERVISION_TIMEOUT.
	HCI_WRITE_LOCAL_NAME	This is macro HCI_WRITE_LOCAL_NAME.
	HCI_WRITE_LOCAL_NAME_PARAM_LEN	This is macro HCI_WRITE_LOCAL_NAME_PARAM_LEN.
	HCI_WRITE_LOOPBACK_MODE	This is macro HCI_WRITE_LOOPBACK_MODE.
	HCI_WRITE_NUM_BROADCST_RETR	This is macro HCI_WRITE_NUM_BROADCST_RETR.
	HCI_WRITE_PAGE_SCAN_ACTIVITY	This is macro HCI_WRITE_PAGE_SCAN_ACTIVITY.
	HCI_WRITE_PAGE_SCAN_PERIOD_MODE	This is macro HCI_WRITE_PAGE_SCAN_PERIOD_MODE.
	HCI_WRITE_PAGE_SCAN_TYPE	This is macro HCI_WRITE_PAGE_SCAN_TYPE.
	HCI_WRITE_PAGE_TIMEOUT	This is macro HCI_WRITE_PAGE_TIMEOUT.
	HCI_WRITE_PIN_TYPE	This is macro HCI_WRITE_PIN_TYPE.
	HCI_WRITE_SCAN_ENABLE	This is macro HCI_WRITE_SCAN_ENABLE.
	HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE.
	HCI_WRITE_SIMPLE_PAIRING_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_MODE.
	HCI_WRITE_STORED_LINK_KEY	This is macro HCI_WRITE_STORED_LINK_KEY.
	HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE.
	HCI_WRITE_VOICE_SETTING	This is macro HCI_WRITE_VOICE_SETTING.
	OGF_CTRL_BASEBAND	This is macro OGF_CTRL_BASEBAND.
	OGF_INFORMATION	This is macro OGF_INFORMATION.
	OGF_LE	This is macro OGF_LE.
	OGF_LINK_CONTROL	This is macro OGF_LINK_CONTROL.
	OGF_LINK_POLICY	This is macro OGF_LINK_POLICY.
	OGF_STATUS	This is macro OGF_STATUS.
	OGF_TESTING	This is macro OGF_TESTING.
	OGF_VENDOR	This is macro OGF_VENDOR.

Structures

	Name	Description
	_bt_hci_command_s	This is type bt_hci_command_t.
	_bt_hci_data_s	This is type bt_hci_data_t.
	_bt_hci_event_s	This is type bt_hci_event_t.
	_bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.
	bt_hci_command_p	This is type bt_hci_command_p.
	bt_hci_command_t	This is type bt_hci_command_t.
	bt_hci_data_p	This is type bt_hci_data_p.
	bt_hci_data_t	This is type bt_hci_data_t.
	bt_hci_event_p	This is type bt_hci_event_p.
	bt_hci_event_t	This is type bt_hci_event_t.
	bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.

Types

	Name	Description
	bt_hci_cmd_callback_fp	This is type bt_hci_cmd_callback_fp.
	bt_hci_conn_state_p	This is type bt_hci_conn_state_p.
	bt_hci_conn_state_t	This is type bt_hci_conn_state_t.
	bt_hci_connect_callback_fp	brief HCI connect callback. ingroup hci details This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete. param status Operation status. c It is 0 if connection was successfully established. param pconn pointer to a structure representing the established connection. param param pointer to arbitrary data passed to the bt_hci_connect() function through its c param parameter.
	bt_hci_data_callback_fp	This is type bt_hci_data_callback_fp.
	bt_hci_disconnect_callback_fp	brief HCI disconnect callback. ingroup hci details This typedef defines a type for the callback function that is called when an HCI connection has been terminated. param status Operation status. c It is 0 if connection has been successfully terminated. param reason Reason for disconnection. param pconn pointer to a structure representing the connection. param param pointer to arbitrary data associated with an event listener.
	bt_hci_event_listener_fp	This is type bt_hci_event_listener_fp.
	bt_hci_hconn_p	This is type bt_hci_hconn_p.
	bt_hci_hconn_t	This is type bt_hci_hconn_t.
	bt_hci_inquiry_callback_fp	This is type bt_hci_inquiry_callback_fp.
	bt_hci_read_inquiry_mode_callback_fp	This is type bt_hci_read_inquiry_mode_callback_fp.
	bt_hci_read_inquiry_scan_activity_callback_fp	This is type bt_hci_read_inquiry_scan_activity_callback_fp.
	bt_hci_read_inquiry_scan_type_callback_fp	This is type bt_hci_read_inquiry_scan_type_callback_fp.
	bt_hci_read_page_scan_activity_callback_fp	This is type bt_hci_read_page_scan_activity_callback_fp.
	bt_hci_read_page_scan_period_mode_callback_fp	This is type bt_hci_read_page_scan_period_mode_callback_fp.
	bt_hci_read_page_scan_type_callback_fp	This is type bt_hci_read_page_scan_type_callback_fp.
	bt_hci_read_page_timeout_callback_fp	This is type bt_hci_read_page_timeout_callback_fp.
	bt_hci_request_remote_name_callback_fp	This is type bt_hci_request_remote_name_callback_fp.
	bt_hci_start_callback_fp	brief HCI initialization callback. ingroup hci details This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function. param success Specifies whether HCI initialization succeeded or not.
	bt_hci_stop_callback_fp	brief HCI stop callback. ingroup hci details This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.
	pf_hci_sleep_callback	This is type pf_hci_sleep_callback.
	pf_hci_wakeup_callback	This is type pf_hci_wakeup_callback.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a

hci_cmd_buffer.h

Functions

	Name	Description
	_hci_allocate_cmd	This is function _hci_allocate_cmd.
	_hci_free_cmd	This is function _hci_free_cmd.
	_hci_init_cmd_buffers	This is function _hci_init_cmd_buffers.

Description

hci_cmd_queue.h

Functions

	Name	Description
	_hci_init_cmd_queues	This is function _hci_init_cmd_queues.
	hci_cq_find_by_bdaddr_and_opcode	This is function hci_cq_find_by_bdaddr_and_opcode.
	hci_cq_find_by_hconn	This is function hci_cq_find_by_hconn.
	hci_cq_find_by_hconn_and_opcode	This is function hci_cq_find_by_hconn_and_opcode.
	hci_cq_find_by_opcode	This is function hci_cq_find_by_opcode.

Description

hci_config.h

Macros

	Name	Description
	__HCI_CONFIG_H	This is macro __HCI_CONFIG_H.
	BT_ENABLE_SCO	This is macro BT_ENABLE_SCO.
	HCI_ALLOCATE_BUFFERS_FUNCTION	This is macro HCI_ALLOCATE_BUFFERS_FUNCTION.
	HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	HCI_ALLOCATE_BUFFERS_VARS	This is macro HCI_ALLOCATE_BUFFERS_VARS.
	HCI_DECLARE_LE_CONN_STATES	This is macro HCI_DECLARE_LE_CONN_STATES.
	HCI_DECLARE_LE_CTRL_STATE	This is macro HCI_DECLARE_LE_CTRL_STATE.
	HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_INIT_LE_CONN_STATES	This is macro HCI_INIT_LE_CONN_STATES.
	HCI_INIT_LE_CTRL_STATE	This is macro HCI_INIT_LE_CTRL_STATE.
	HCI_INIT_SCO_HANDLERS	This is macro HCI_INIT_SCO_HANDLERS.
	HCI_INIT_SSP_HANDLERS	This is macro HCI_INIT_SSP_HANDLERS.
	HCI_L2CAP_BUFFER_LEN	This is macro HCI_L2CAP_BUFFER_LEN.
	HCI_MAX_CONNECT_ATTEMPTS	This is macro HCI_MAX_CONNECT_ATTEMPTS.
	HCI_SIZEOF_LE_CONN_STATES	This is macro HCI_SIZEOF_LE_CONN_STATES.

	HCI_SIZEOF_LE_CTRL_STATE	This is macro HCI_SIZEOF_LE_CTRL_STATE.
	HCI_TX_BUFFER_LEN	This is macro HCI_TX_BUFFER_LEN.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_config_event_handlers.h

Macros

	Name	Description
	__HCI_CONFIG_EVENT_HANDLERS_H	This is macro __HCI_CONFIG_EVENT_HANDLERS_H.
	BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER.
	BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER.
	BT_HCI_EVT_CONNECTION_REQUEST_HANDLER	This is macro BT_HCI_EVT_CONNECTION_REQUEST_HANDLER.
	BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER	This is macro BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER.
	BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER	This is macro BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER.
	BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER.
	BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER.
	BT_HCI_EVT_MODE_CHANGE_HANDLER	This is macro BT_HCI_EVT_MODE_CHANGE_HANDLER.
	BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER	This is macro BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER.
	BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER	This is macro BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER.
	BT_HCI_EVT_ROLE_CHANGE_HANDLER	This is macro BT_HCI_EVT_ROLE_CHANGE_HANDLER.
	BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER.
	BT_LE_EVT_HANDLER	This is macro BT_LE_EVT_HANDLER.
	BT_SSP_EVT_HANDLER	This is macro BT_SSP_EVT_HANDLER.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_conn_state.h

Functions

	Name	Description
	bt_hci_register_listener	This is function bt_hci_register_listener.
	bt_hci_unregister_listener	This is function bt_hci_unregister_listener.
	hci_allocate_conn_state	This is function hci_allocate_conn_state.
	hci_get_conn_state	This is function hci_get_conn_state.
	hci_get_conn_state_by_bdaddr	This is function hci_get_conn_state_by_bdaddr.

Macros

	Name	Description
	HCI_CONN_ROLE_MASTER	This is macro HCI_CONN_ROLE_MASTER.
	HCI_CONN_ROLE_SLAVE	This is macro HCI_CONN_ROLE_SLAVE.
	HCI_CONN_STATE_AUTHENTICATING	This is macro HCI_CONN_STATE_AUTHENTICATING.
	HCI_CONN_STATE_CLOSED	This is macro HCI_CONN_STATE_CLOSED.
	HCI_CONN_STATE_OPEN	This is macro HCI_CONN_STATE_OPEN.
	HCI_CONN_TYPE_ACL	This is macro HCI_CONN_TYPE_ACL.
	HCI_CONN_TYPE_ESCO	This is macro HCI_CONN_TYPE_ESCO.
	HCI_CONN_TYPE_SCO	This is macro HCI_CONN_TYPE_SCO.
	HCI_LINK_TYPE_BD_EDR	This is macro HCI_LINK_TYPE_BD_EDR.
	HCI_LINK_TYPE_LE	This is macro HCI_LINK_TYPE_LE.

Structures

	Name	Description
	_bt_hci_conn_state_s	This is record _bt_hci_conn_state_s.
	_bt_hci_listener_t	This is record _bt_hci_listener_t.

Types

	Name	Description
	bt_hci_listener_t	This is type bt_hci_listener_t.
	bt_hci_sco_read_data_callback_fp	This is type bt_hci_sco_read_data_callback_fp.

Unions

	Name	Description
	_bt_hci_event_e	This is type bt_hci_event_e.
	bt_hci_event_e	This is type bt_hci_event_e.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_ctrl_state.h

Functions

	Name	Description
	bt_hci_ctrl_register_data_listener	This is function bt_hci_ctrl_register_data_listener.
	bt_hci_ctrl_register_listener	This is function bt_hci_ctrl_register_listener.
	bt_hci_ctrl_unregister_listener	This is function bt_hci_ctrl_unregister_listener.
	bt_hci_set_event_listener	listener is triggered by the following events: HCI_EVT_AUTHENTICATION_COMPLETE HCI_EVT_CONNECTION_COMPLETE HCI_EVT_DISCONNECTION_COMPLETE HCI_EVT_ROLE_CHANGE HCI_EVT_MODE_CHANGE

Macros

	Name	Description
	HCI_CTRL_LISTENER_ACL_DATA	This is macro HCI_CTRL_LISTENER_ACL_DATA.
	HCI_CTRL_LISTENER_EVENT	This is macro HCI_CTRL_LISTENER_EVENT.
	HCI_CTRL_LISTENER_SCO_DATA	This is macro HCI_CTRL_LISTENER_SCO_DATA.
	HCI_CTRL_STATE_CLOSED	This is macro HCI_CTRL_STATE_CLOSED.
	HCI_CTRL_STATE_CONNECTABLE	This is macro HCI_CTRL_STATE_CONNECTABLE.
	HCI_CTRL_STATE_DISCOVERABLE	This is macro HCI_CTRL_STATE_DISCOVERABLE.
	HCI_CTRL_STATE_INIT	This is macro HCI_CTRL_STATE_INIT.
	HCI_CTRL_STATE_READY	This is macro HCI_CTRL_STATE_READY.
	HCI_CTRL_STATE_SLEEP	This is macro HCI_CTRL_STATE_SLEEP.
	HCI_CTRL_STATE_WAKING_UP	This is macro HCI_CTRL_STATE_WAKING_UP.

Structures

	Name	Description
	_bt_hci_ctrl_listener_t	This is record _bt_hci_ctrl_listener_t.
	_bt_hci_ctrl_state_s	This is type bt_hci_ctrl_state_t.
	bt_hci_ctrl_state_t	This is type bt_hci_ctrl_state_t.

Types

	Name	Description
	bt_hci_cmd_listener_fp	This is type bt_hci_cmd_listener_fp.
	bt_hci_ctrl_listener_t	This is type bt_hci_ctrl_listener_t.
	bt_hci_data_listener_fp	This is type bt_hci_data_listener_fp.
	pf_l2cap_receive_callback	This is type pf_l2cap_receive_callback.

Description

hci_data_buffer.h

Functions

	Name	Description
	bt_hci_alloc_data_buffer	This is function bt_hci_alloc_data_buffer.
	bt_hci_free_data_buffer	This is function bt_hci_free_data_buffer.
	bt_hci_init_data_buffers	This is function bt_hci_init_data_buffers.

Macros

	Name	Description
	HCI_DATA_BUFFER_STATE_FREE	This is macro HCI_DATA_BUFFER_STATE_FREE.
	HCI_DATA_BUFFER_STATE_USED	This is macro HCI_DATA_BUFFER_STATE_USED.
	HCI_MAX_DATA_BUFFERS	This is macro HCI_MAX_DATA_BUFFERS.

Structures

	Name	Description
	_bt_hci_data_buffer_s	This is type bt_hci_data_buffer_t.
	bt_hci_data_buffer_p	This is type bt_hci_data_buffer_p.
	bt_hci_data_buffer_t	This is type bt_hci_data_buffer_t.

Description

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

hci_data_queue.h

Functions

	Name	Description
	bt_hci_init_data_queues	This is function bt_hci_init_data_queues.

Description

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

hci_eir.h

Functions

	Name	Description
	bt_hci_allocate_write_eir_command	This is function bt_hci_allocate_write_eir_command.
	bt_hci_param_eir_add	This is function bt_hci_param_eir_add.
	bt_hci_param_eir_device_id_add	This is function bt_hci_param_eir_device_id_add.
	bt_hci_param_eir_local_name_add	This is function bt_hci_param_eir_local_name_add.
	bt_hci_param_eir_uuid128_add	This is function bt_hci_param_eir_uuid128_add.
	bt_hci_param_eir_uuid16_add	This is function bt_hci_param_eir_uuid16_add.
	bt_hci_param_eir_uuid32_add	This is function bt_hci_param_eir_uuid32_add.
	bt_hci_param_eir_vendor_add	This is function bt_hci_param_eir_vendor_add.
	bt_hci_param_tx_power_level_add	This is function bt_hci_param_tx_power_level_add.
	bt_hci_write_eir	This is function bt_hci_write_eir.

Macros

	Name	Description
	HCI_EIR_FEC_NOT_REQUIRED	This is macro HCI_EIR_FEC_NOT_REQUIRED.
	HCI_EIR_FEC_REQUIRED	This is macro HCI_EIR_FEC_REQUIRED.
	HCI_EIR_TYPE_3D_Information_Data	This is macro HCI_EIR_TYPE_3D_Information_Data.
	HCI_EIR_TYPE_ADVERTISING_INTERVAL	This is macro HCI_EIR_TYPE_ADVERTISING_INTERVAL.
	HCI_EIR_TYPE_APPEARANCE	This is macro HCI_EIR_TYPE_APPEARANCE.
	HCI_EIR_TYPE_DEVICE_ID	This is macro HCI_EIR_TYPE_DEVICE_ID.
	HCI_EIR_TYPE_FLAGS	This is macro HCI_EIR_TYPE_FLAGS.
	HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS	This is macro HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS.
	HCI_EIR_TYPE_LE_ROLE	This is macro HCI_EIR_TYPE_LE_ROLE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE.
	HCI_EIR_TYPE_LOCAL_NAME_COMPLETE	This is macro HCI_EIR_TYPE_LOCAL_NAME_COMPLETE.
	HCI_EIR_TYPE_LOCAL_NAME_SHORTENED	This is macro HCI_EIR_TYPE_LOCAL_NAME_SHORTENED.
	HCI_EIR_TYPE_MANUFACTURER_SPECIFIC	This is macro HCI_EIR_TYPE_MANUFACTURER_SPECIFIC.

	HCI_EIR_TYPE_OOB_COD	This is macro HCI_EIR_TYPE_OOB_COD.
	HCI_EIR_TYPE_OOB_HASH	This is macro HCI_EIR_TYPE_OOB_HASH.
	HCI_EIR_TYPE_OOB_RANDOMIZER	This is macro HCI_EIR_TYPE_OOB_RANDOMIZER.
	HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS.
	HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS.
	HCI_EIR_TYPE_SERVICE_DATA	This is macro HCI_EIR_TYPE_SERVICE_DATA.
	HCI_EIR_TYPE_SERVICE_DATA_UUID128	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID128.
	HCI_EIR_TYPE_SERVICE_DATA_UUID16	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID16.
	HCI_EIR_TYPE_SERVICE_DATA_UUID32	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID32.
	HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256.
	HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256.
	HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE	This is macro HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE.
	HCI_EIR_TYPE_SM_OOB_FLAGS	This is macro HCI_EIR_TYPE_SM_OOB_FLAGS.
	HCI_EIR_TYPE_SM_TK_VALUE	This is macro HCI_EIR_TYPE_SM_TK_VALUE.
	HCI_EIR_TYPE_SOLICITATION_UUID128_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID128_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID16_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID16_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID32_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID32_LIST.
	HCI_EIR_TYPE_TX_POWER_LEVEL	This is macro HCI_EIR_TYPE_TX_POWER_LEVEL.
	HCI_EIR_TYPE_UUID128_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID128_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID16_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID16_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID32_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID32_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE.

Description

hci_errors.h

Macros

	Name	Description
	HCI_ERR_ACL_CONN_ALREADY_EXISTS	This is macro HCI_ERR_ACL_CONN_ALREADY_EXISTS.
	HCI_ERR_AUTHENTICATION_FAILURE	This is macro HCI_ERR_AUTHENTICATION_FAILURE.
	HCI_ERR_CONN_REJECT_LIMITED_RESOURCES	This is macro HCI_ERR_CONN_REJECT_LIMITED_RESOURCES.
	HCI_ERR_CONNECTION_TIMEOUT	This is macro HCI_ERR_CONNECTION_TIMEOUT.

	HCI_ERR_INVALID_PARAMETERS	This is macro HCI_ERR_INVALID_PARAMETERS.
	HCI_ERR_MEMORY_CAPACITY_EXCEEDED	This is macro HCI_ERR_MEMORY_CAPACITY_EXCEEDED.
	HCI_ERR_PAIRING_NOT_ALLOWED	This is macro HCI_ERR_PAIRING_NOT_ALLOWED.
	HCI_ERR_PIN_OR_KEY_MISSING	This is macro HCI_ERR_PIN_OR_KEY_MISSING.
	HCI_ERR_SCO_CONN_LIMIT_EXCEEDED	This is macro HCI_ERR_SCO_CONN_LIMIT_EXCEEDED.
	HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED	This is macro HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED.
	HCI_ERR_SUCCESS	• addtogroup hci • @{ • @name Errors
	HCI_ERR_UNSPECIFIED	This is macro HCI_ERR_UNSPECIFIED.
	HCI_SUCCESS	This is macro HCI_SUCCESS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_evt_handlers.h

Functions

	Name	Description
	bt_hci_evt_authentication_complete_handler	This is function bt_hci_evt_authentication_complete_handler.
	bt_hci_evt_change_conn_link_complete_handler	This is function bt_hci_evt_change_conn_link_complete_handler.
	bt_hci_evt_command_complete_handler	This is function bt_hci_evt_command_complete_handler.
	bt_hci_evt_command_status_handler	This is function bt_hci_evt_command_status_handler.
	bt_hci_evt_conn_packet_type_changed_handler	This is function bt_hci_evt_conn_packet_type_changed_handler.
	bt_hci_evt_connection_complete_handler	This is function bt_hci_evt_connection_complete_handler.
	bt_hci_evt_connection_request_handler	This is function bt_hci_evt_connection_request_handler.
	bt_hci_evt_data_buffer_overflow_handler	This is function bt_hci_evt_data_buffer_overflow_handler.
	bt_hci_evt_default_handler	This is function bt_hci_evt_default_handler.
	bt_hci_evt_disconnection_complete_handler	This is function bt_hci_evt_disconnection_complete_handler.
	bt_hci_evt_encryption_change_handler	This is function bt_hci_evt_encryption_change_handler.
	bt_hci_evt_extended_inquiry_result_handler	This is function bt_hci_evt_extended_inquiry_result_handler.
	bt_hci_evt_flow_specification_complete_handler	This is function bt_hci_evt_flow_specification_complete_handler.
	bt_hci_evt_flush_occured_handler	This is function bt_hci_evt_flush_occured_handler.
	bt_hci_evt_hardware_error_handler	This is function bt_hci_evt_hardware_error_handler.
	bt_hci_evt_inquiry_complete_handler	This is function bt_hci_evt_inquiry_complete_handler.
	bt_hci_evt_inquiry_result_handler	This is function bt_hci_evt_inquiry_result_handler.
	bt_hci_evt_inquiry_result_with_rssi_handler	This is function bt_hci_evt_inquiry_result_with_rssi_handler.
	bt_hci_evt_link_key_notification_handler	This is function bt_hci_evt_link_key_notification_handler.
	bt_hci_evt_link_key_request_handler	This is function bt_hci_evt_link_key_request_handler.
	bt_hci_evt_loopback_command_handler	This is function bt_hci_evt_loopback_command_handler.
	bt_hci_evt_master_link_key_complete_handler	This is function bt_hci_evt_master_link_key_complete_handler.
	bt_hci_evt_max_slots_change_handler	This is function bt_hci_evt_max_slots_change_handler.
	bt_hci_evt_mode_change_handler	This is function bt_hci_evt_mode_change_handler.
	bt_hci_evt_num_of_completed_packets_handler	This is function bt_hci_evt_num_of_completed_packets_handler.
	bt_hci_evt_page_scan_repet_mode_change_handler	This is function bt_hci_evt_page_scan_repet_mode_change_handler.
	bt_hci_evt_pin_code_request_handler	This is function bt_hci_evt_pin_code_request_handler.
	bt_hci_evt_qos_setup_complete_handler	This is function bt_hci_evt_qos_setup_complete_handler.
	bt_hci_evt_qos_violation_handler	This is function bt_hci_evt_qos_violation_handler.
	bt_hci_evt_read_clock_offset_complete_handler	This is function bt_hci_evt_read_clock_offset_complete_handler.
	bt_hci_evt_read_rmt_ext_features_comp_handler	This is function bt_hci_evt_read_rmt_ext_features_comp_handler.
	bt_hci_evt_read_rmt_sup_features_comp_handler	This is function bt_hci_evt_read_rmt_sup_features_comp_handler.

	bt_hci_evt_read_rmt_version_info_comp_handler	This is function bt_hci_evt_read_rmt_version_info_comp_handler.
	bt_hci_evt_remote_name_request_complete_handler	This is function bt_hci_evt_remote_name_request_complete_handler.
	bt_hci_evt_return_link_keys_handler	This is function bt_hci_evt_return_link_keys_handler.
	bt_hci_evt_role_change_handler	This is function bt_hci_evt_role_change_handler.
	bt_hci_evt_synch_connection_changed_handler	This is function bt_hci_evt_synch_connection_changed_handler.
	bt_hci_evt_synch_connection_complete_handler	This is function bt_hci_evt_synch_connection_complete_handler.
	bt_hci_set_vendor_specific_event_handler	This is function bt_hci_set_vendor_specific_event_handler.

Structures

	Name	Description
	_bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	_bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	_bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	_bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	_bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	_bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	_bt_hci_evt_encryption_change_s	This is type bt_hci_evt_encryption_change_t.
	_bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	_bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.
	bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	bt_hci_evt_encryption_change_t	This is type bt_hci_evt_encryption_change_t.
	bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.

Types

	Name	Description
	bt_hci_event_handler_ex_fp	This is type bt_hci_event_handler_ex_fp.
	bt_hci_event_handler_fp	This is type bt_hci_event_handler_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_le.h

Functions

	Name	Description
	bt_hci_le_add_device_to_white_list	This is function bt_hci_le_add_device_to_white_list.
	bt_hci_le_advertising_add	This is function bt_hci_le_advertising_add.
	bt_hci_le_advertising_add_local_name	This is function bt_hci_le_advertising_add_local_name.
	bt_hci_le_advertising_device_id_add	This is function bt_hci_le_advertising_device_id_add.
	bt_hci_le_advertising_flags_add	This is function bt_hci_le_advertising_flags_add.
	bt_hci_le_advertising_get_local_name	This is function bt_hci_le_advertising_get_local_name.
	bt_hci_le_advertising_tx_power_level_add	This is function bt_hci_le_advertising_tx_power_level_add.
	bt_hci_le_advertising_uuid128_add	This is function bt_hci_le_advertising_uuid128_add.
	bt_hci_le_advertising_uuid16_add	This is function bt_hci_le_advertising_uuid16_add.
	bt_hci_le_advertising_uuid32_add	This is function bt_hci_le_advertising_uuid32_add.

	bt_hci_le_advertising_vendor_add	This is function bt_hci_le_advertising_vendor_add.
	bt_hci_le_allocate_set_advertising_data_command	This is function bt_hci_le_allocate_set_advertising_data_command.
	bt_hci_le_allocate_set_scan_response_data_command	This is function bt_hci_le_allocate_set_scan_response_data_command.
	bt_hci_le_cancel_connect_ex	This is function bt_hci_le_cancel_connect_ex.
	bt_hci_le_cancel_find_devices	This is function bt_hci_le_cancel_find_devices.
	bt_hci_le_clear_white_list	This is function bt_hci_le_clear_white_list.
	bt_hci_le_connect_ex	This is function bt_hci_le_connect_ex.
	bt_hci_le_enable	This is function bt_hci_le_enable.
	bt_hci_le_encrypt	This is function bt_hci_le_encrypt.
	bt_hci_le_find_devices	This is function bt_hci_le_find_devices.
	bt_hci_le_get_connect_parameters	This is function bt_hci_le_get_connect_parameters.
	bt_hci_le_ibeacon_add	This is function bt_hci_le_ibeacon_add.
	bt_hci_le_ltk_negative_reply	This is function bt_hci_le_ltk_negative_reply.
	bt_hci_le_ltk_reply	This is function bt_hci_le_ltk_reply.
	bt_hci_le_rand	This is function bt_hci_le_rand.
	bt_hci_le_read_channel_map	This is function bt_hci_le_read_channel_map.
	bt_hci_le_read_remote_used_features	This is function bt_hci_le_read_remote_used_features.
	bt_hci_le_read_support	This is function bt_hci_le_read_support.
	bt_hci_le_read_white_list_size	This is function bt_hci_le_read_white_list_size.
	bt_hci_le_receiver_test	This is function bt_hci_le_receiver_test.
	bt_hci_le_remove_device_from_white_list	This is function bt_hci_le_remove_device_from_white_list.
	bt_hci_le_set_adevrtising_enable_ex	This is function bt_hci_le_set_adevrtising_enable_ex.
	bt_hci_le_set_advertising_parameters	This is function bt_hci_le_set_advertising_parameters.
	bt_hci_le_set_connect_parameters	This is function bt_hci_le_set_connect_parameters.
	bt_hci_le_set_host_channel_classification	This is function bt_hci_le_set_host_channel_classification.
	bt_hci_le_set_random_address	This is function bt_hci_le_set_random_address.
	bt_hci_le_set_scan_enable	This is function bt_hci_le_set_scan_enable.
	bt_hci_le_set_scan_parameters	This is function bt_hci_le_set_scan_parameters.
	bt_hci_le_start_encryption	This is function bt_hci_le_start_encryption.
	bt_hci_le_supported	This is function bt_hci_le_supported.
	bt_hci_le_test_end	This is function bt_hci_le_test_end.
	bt_hci_le_transmitter_test	This is function bt_hci_le_transmitter_test.
	bt_hci_le_update_connection	This is function bt_hci_le_update_connection.
	bt_hci_le_write_support	This is function bt_hci_le_write_support.

Macros

	Name	Description
	bt_hci_le_cancel_connect	This is macro bt_hci_le_cancel_connect.
	bt_hci_le_set_adevrtising_enable	This is macro bt_hci_le_set_adevrtising_enable.
	HCI_LE_ADDRESS_TYPE_PUBLIC	This is macro HCI_LE_ADDRESS_TYPE_PUBLIC.
	HCI_LE_ADDRESS_TYPE_RANDOM	This is macro HCI_LE_ADDRESS_TYPE_RANDOM.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_37	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_37.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_38	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_38.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_39	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_39.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL.
	HCI_LE_ADV_CHANNEL_MAP_RESERVED	This is macro HCI_LE_ADV_CHANNEL_MAP_RESERVED.
	HCI_LE_ADV_TYPE_CONN_UNDIRECT	Connectable undirected advertising (ADV_IND) (default)
	HCI_LE_ADV_TYPE_DIRECT_HIGH	Connectable high duty cycle directed advertising (ADV_DIRECT_IND, high duty cycle)

	HCI_LE_ADV_TYPE_DIRECT_LOW	Connectable low duty cycle directed advertising (ADV_DIRECT_IND, low duty cycle)
	HCI_LE_ADV_TYPE_NONCONN	Non connectable undirected advertising (ADV_NONCONN_IND)
	HCI_LE_ADV_TYPE_SCAN	Scannable undirected advertising (ADV_SCAN_IND)
	HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED	This is macro HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED.
	HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST.
	HCI_LE_AVERTISING_DISABLED	This is macro HCI_LE_AVERTISING_DISABLED.
	HCI_LE_AVERTISING_ENABLED	This is macro HCI_LE_AVERTISING_ENABLED.
	HCI_LE_DISABLED	This is macro HCI_LE_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_DISABLED	This is macro HCI_LE_DUPLICATE_FILTERING_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_ENABLED	This is macro HCI_LE_DUPLICATE_FILTERING_ENABLED.
	HCI_LE_ENABLED	This is macro HCI_LE_ENABLED.
	HCI_LE_EVT_ADVERTISING_REPORT	This is macro HCI_LE_EVT_ADVERTISING_REPORT.
	HCI_LE_EVT_CONNECTION_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_COMPLETE.
	HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE.
	HCI_LE_EVT_LONG_TERM_KEY_REQUEST	This is macro HCI_LE_EVT_LONG_TERM_KEY_REQUEST.
	HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE	This is macro HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE.
	HCI_LE_FILTER_POLICY_NOT_USED	White list is not used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be used.
	HCI_LE_FILTER_POLICY_WHITE_LIST	White list is used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be ignored
	HCI_LE_FLAG_SHARED_ACL_BUFFERS	This is macro HCI_LE_FLAG_SHARED_ACL_BUFFERS.
	HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR	This is macro HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR.
	HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_STATIC	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_STATIC.
	HCI_LE_SCAN_DISABLED	This is macro HCI_LE_SCAN_DISABLED.
	HCI_LE_SCAN_ENABLED	This is macro HCI_LE_SCAN_ENABLED.
	HCI_LE_SCAN_EVT_CANCEL_FAILED	This is macro HCI_LE_SCAN_EVT_CANCEL_FAILED.

	HCI_LE_SCAN_EVT_CANCELLED	This is macro HCI_LE_SCAN_EVT_CANCELLED.
	HCI_LE_SCAN_EVT_DEVICE_FOUND	This is macro HCI_LE_SCAN_EVT_DEVICE_FOUND.
	HCI_LE_SCAN_EVT_START_FAILED	This is macro HCI_LE_SCAN_EVT_START_FAILED.
	HCI_LE_SCAN_EVT_STARTED	This is macro HCI_LE_SCAN_EVT_STARTED.
	HCI_LE_SCAN_FILTER_POLICY_ALL	This is macro HCI_LE_SCAN_FILTER_POLICY_ALL.
	HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST	This is macro HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST.
	HCI_LE_SCAN_TYPE_ACTIVE	This is macro HCI_LE_SCAN_TYPE_ACTIVE.
	HCI_LE_SCAN_TYPE_PASSIVE	This is macro HCI_LE_SCAN_TYPE_PASSIVE.
	HCI_LE_SIMULTANEOUS_DISABLED	This is macro HCI_LE_SIMULTANEOUS_DISABLED.
	HCI_LE_SIMULTANEOUS_ENABLED	This is macro HCI_LE_SIMULTANEOUS_ENABLED.

Structures

	Name	Description
	_bt_hci_le_advertising_report_t	This is record _bt_hci_le_advertising_report_t.
	_bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	_bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	_bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	_bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	_bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	_bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.
	bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.

Types

	Name	Description
	bt_hci_le_advertising_report_t	This is type bt_hci_le_advertising_report_t.
	bt_hci_le_scan_callback_fp	This is type bt_hci_le_scan_callback_fp.
	bt_le_evt_handler	This is type bt_le_evt_handler.

Description

hci_linkkey_buffer.h

Functions

	Name	Description
	bt_hci_init_linkkey_buffers	This is function bt_hci_init_linkkey_buffers.

Structures

	Name	Description
	_bt_hci_link_key_s	This is type bt_hci_link_key_t.
	bt_hci_link_key_t	This is type bt_hci_link_key_t.

Description

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

hci_private.h

Functions

	Name	Description
	_bt_hci_ctrl_notify_data_listeners	This is function _bt_hci_ctrl_notify_data_listeners.
	_bt_hci_ctrl_notify_listeners	This is function _bt_hci_ctrl_notify_listeners.
	_bt_hci_get_tick_count	This is function _bt_hci_get_tick_count.
	_bt_hci_init_timer	From hci_timer.c
	_bt_hci_init_transport	This is function _bt_hci_init_transport.
	_bt_hci_le_command_complete_handler	This is function _bt_hci_le_command_complete_handler.
	_bt_hci_notify_listeners	This is function _bt_hci_notify_listeners.
	_bt_hci_set_init_flags	This is function _bt_hci_set_init_flags.
	_bt_le_evt_handler	This is function _bt_le_evt_handler.
	_hci_allocate_buffers	Defined by OEM through library configuration
	_hci_receive_start	From hci_receive.c
	_hci_recv_sco_data_packet	From hci_sco.c
	_hci_send_commands_from_queue	From hci_send.c
	_hci_send_data	This is function _hci_send_data.
	_hci_send_data_fragment	This is function _hci_send_data_fragment.
	_hci_send_data_from_queue	This is function _hci_send_data_from_queue.
	bt_hci_le_init	This is function bt_hci_le_init.
	hci_check_aux_info	From hci_aux_info.c

Macros

	Name	Description
	HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS	The total number of ACL data packets read from the controller will be ignored. The stack will assume that he controller can accept only 1 ACL packet and sends HCI_EVT_NUM_OF_COMPLETED_PACKETS for each packet it has processed. This seems to be for controller working over SDIO (at least for Marvell's 88W8777).
	HCI_INIT_FLAG_SEND_HCI_RESET	HCI reset will be sent when bt_hci_init is called

Variables

	Name	Description
	_bt_hci_le_init	This is variable _bt_hci_le_init.
	_conn_state_recv_buffers	This is variable _conn_state_recv_buffers.
	_hci_cmd_buffer_headers	Global variables defined by OEM configuration
	_hci_cmd_buffers	This is variable _hci_cmd_buffers.
	_hci_cmd_mgr	In hci_cmd_buffer.c
	_hci_cmd_param_buffers	This is variable _hci_cmd_param_buffers.
	_hci_connections	This is variable _hci_connections.
	_hci_enable_ctrl_to_host_flow_control	This is variable _hci_enable_ctrl_to_host_flow_control.
	_hci_enable_sco	This is variable _hci_enable_sco.
	_hci_event_handlers	This is variable _hci_event_handlers.
	_hci_evt_synch_connection_complete_handler	This is variable _hci_evt_synch_connection_complete_handler.
	_hci_l2cap_buffer_len	This is variable _hci_l2cap_buffer_len.
	_hci_linkkey_mgr	In hci_linkkey_buffer.c
	_hci_max_cmd_buffers	This is variable _hci_max_cmd_buffers.
	_hci_max_cmd_param_len	This is variable _hci_max_cmd_param_len.
	_hci_max_connect_attempts	This is variable _hci_max_connect_attempts.
	_hci_max_data_buffers	This is variable _hci_max_data_buffers.
	_hci_max_hci_connections	This is variable _hci_max_hci_connections.

	_hci_rcv_buffer_len	This is variable _hci_rcv_buffer_len.
	_hci_recv_sco_data_packet_fp	This is variable _hci_recv_sco_data_packet_fp.
	_hci_send_data_buffer_headers	This is variable _hci_send_data_buffer_headers.
	_hci_send_data_buffers	This is variable _hci_send_data_buffers.
	_hci_send_data_mgr	In hci_data_buffer.c
	_hci_tx_buffer_len	This is variable _hci_tx_buffer_len.
	_pcmd_being_sent	In hci_send.c
	_pdata_being_sent	This is variable _pdata_being_sent.
	_phci_ctrl	In hci_param.c
	_ram_size_hci_buffers	This is variable _ram_size_hci_buffers.
	_ram_size_hci_cmd_queue	This is variable _ram_size_hci_cmd_queue.
	_ram_size_hci_linkkey_buffer	This is variable _ram_size_hci_linkkey_buffer.
	_ram_size_hci_param	This is variable _ram_size_hci_param.
	_ram_size_linkkey_storage	This is variable _ram_size_linkkey_storage.
	_recv_buffer	This is variable _recv_buffer.
	_resp_cq_head	This is variable _resp_cq_head.
	_send_buffer	This is variable _send_buffer.
	_send_cq_head	In hci_cmd_queue.c

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_signal.h

Functions

	Name	Description
	_bt_hci_init_signal	This is function _bt_hci_init_signal.
	_bt_hci_set_signal	This is function _bt_hci_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_transport.h

Functions

	Name	Description
	bt_hci_transport_recv_packet	This is function bt_hci_transport_recv_packet.
	bt_hci_transport_send_cmd	deprecated
	bt_hci_transport_send_data	This is function bt_hci_transport_send_data.
	bt_hci_transport_send_packet	This is function bt_hci_transport_send_packet.
	bt_hci_transport_set_transport	This is function bt_hci_transport_set_transport.

Macros

	Name	Description
	HCI_TRANSPORT_HEADER_LEN	This is macro HCI_TRANSPORT_HEADER_LEN.

Structures

	Name	Description
	_hci_transport_t	This is type hci_transport_t.
	hci_transport_t	This is type hci_transport_t.

Types

	Name	Description
	bt_hci_transport_recv_packet_callback_fp	This is type bt_hci_transport_recv_packet_callback_fp.
	bt_hci_transport_send_packet_callback_fp	This is type bt_hci_transport_send_packet_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_utils.h

Functions

	Name	Description
	is_bdaddr_command	This is function is_bdaddr_command.
	is_hconn_command	This is function is_hconn_command.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_3wire.h

Functions

	Name	Description
	bt_hcitr_3wire_cancel_recv_packet	This is function bt_hcitr_3wire_cancel_recv_packet.
	bt_hcitr_3wire_init_ex	This is function bt_hcitr_3wire_init_ex.
	bt_hcitr_3wire_reset_ex	This is function bt_hcitr_3wire_reset_ex.
	bt_hcitr_3wire_start	This is function bt_hcitr_3wire_start.

Macros

	Name	Description
	bt_hcitr_3wire_init	This is macro bt_hcitr_3wire_init.
	bt_hcitr_3wire_reset	This is macro bt_hcitr_3wire_reset.
	HCITR_3WIRE_DEFAULT_ACK_TIMEOUT	This is macro HCITR_3WIRE_DEFAULT_ACK_TIMEOUT.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_bcsp.h

Functions

	Name	Description
	bt_hcitr_bcsp_init_ex	This is function bt_hcitr_bcsp_init_ex.
	bt_hcitr_bcsp_reset_ex	This is function bt_hcitr_bcsp_reset_ex.
	bt_hcitr_bcsp_start	This is function bt_hcitr_bcsp_start.

Macros

	Name	Description
	bt_hcitr_bcsp_init	This is macro bt_hcitr_bcsp_init.
	bt_hcitr_bcsp_reset	This is macro bt_hcitr_bcsp_reset.
	HCITR_BCSP_DEFAULT_ACK_TIMEOUT	This is macro HCITR_BCSP_DEFAULT_ACK_TIMEOUT.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_packet.h

Functions

	Name	Description
	bt_hcitr_packet_init	This is function bt_hcitr_packet_init.
	bt_hcitr_packet_reset	This is function bt_hcitr_packet_reset.
	bt_hcitr_packet_start	This is function bt_hcitr_packet_start.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_tih4.h

Enumerations

	Name	Description
	_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.
	bt_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.

Functions

	Name	Description
	bt_hcitr_tih4_init	This is function bt_hcitr_tih4_init.
	bt_hcitr_tih4_reset	This is function bt_hcitr_tih4_reset.
	bt_hcitr_tih4_start	This is function bt_hcitr_tih4_start.
	bt_hcitr_tih4_wake_up	This is function bt_hcitr_tih4_wake_up.

Types

	Name	Description
	bt_hcitr_tih4_power_callback_fp	This is type bt_hcitr_tih4_power_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_uart.h

Functions

	Name	Description
	bt_hcitr_uart_init	brief Initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.
	bt_hcitr_uart_reset	brief Re-initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.
	bt_hcitr_uart_start	brief Start HCI UART (H4) transport protocol ingroup hcitr_uart details This function starts the transport, i.e., makes it able to receive and send packets.

Macros

	Name	Description
	HCI_UART_PACKET_TYPE_ACL_DATA	This is macro HCI_UART_PACKET_TYPE_ACL_DATA.
	HCI_UART_PACKET_TYPE_COMMAND	defgroup hcitr_uart HCI UART (H4) transport protocol ingroup hcitr details This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application: li bt_oem_send() li bt_oem_recv()
	HCI_UART_PACKET_TYPE_EVENT	This is macro HCI_UART_PACKET_TYPE_EVENT.
	HCI_UART_PACKET_TYPE_SCO_DATA	This is macro HCI_UART_PACKET_TYPE_SCO_DATA.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

lm.h

Macros

	Name	Description
	LM_PACKET_TYPE_DH1	This is macro LM_PACKET_TYPE_DH1.
	LM_PACKET_TYPE_DH3	This is macro LM_PACKET_TYPE_DH3.
	LM_PACKET_TYPE_DH5	This is macro LM_PACKET_TYPE_DH5.
	LM_PACKET_TYPE_DM1	This is macro LM_PACKET_TYPE_DM1.
	LM_PACKET_TYPE_DM3	This is macro LM_PACKET_TYPE_DM3.
	LM_PACKET_TYPE_DM5	This is macro LM_PACKET_TYPE_DM5.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap.h

Functions

	Name	Description
	bt_l2cap_allocate_mgr	brief Allocate L2CAP manager. ingroup l2cap details This function allocates and initializes an L2CAP manager structure. One L2CAP manager manages all L2CAP connections for a particular local device. The local device is specified by the c hci_ctrl parameter. param hci_ctrl Pointer to the hci_ctrl_state structure that represents the local device (HCI controller) for which a L2CAP manager is to be allocated. return A pointer to the allocated L2CAP manager structure. The returned L2CAP manager should be freed by a call to bt_l2cap_free_mgr when it is no longer needed.
	bt_l2cap_connect_ext	brief Connect to a remote device. ingroup l2cap details This function establishes an L2CAP connection with a remote device on a specific PSM. When connect operation completes a callback function is called. param mgr The L2CAP manager. param remote_addr The address of the remote device. param psm The PSM for the connection. param connect_cb The Callback function that is called when the connect operation completes. param param A pointer to arbitrary data to be passed to the c connect_cb callback. param state_cb The callback function that is called when the state of the established connection changes. return li c TRUE... more
	bt_l2cap_connect_fixed_channel	This is function bt_l2cap_connect_fixed_channel.
	bt_l2cap_disconnect	brief Close L2CAP channel. ingroup l2cap details This function closes an L2CAP channel. The channel can be established either by a call to bt_l2cap_connect() or by an incoming connection request. param ch The L2CAP channel to be closed. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_disconnect_ex	This is function bt_l2cap_disconnect_ex.
	bt_l2cap_echo	brief Send echo command
	bt_l2cap_free_mgr	brief Release L2CAP manger. ingroup l2cap details This function releases the L2CAP manager structure. param mgr The L2CAP manager structure to be released.
	bt_l2cap_get_mgr	This is function bt_l2cap_get_mgr.
	bt_l2cap_hci_has_open_channels	This is function bt_l2cap_hci_has_open_channels.
	bt_l2cap_init	brief Initialize the L2CAP layer. ingroup l2cap details This function initializes the L2CAP layer of the stack. It must be called prior to any other L2CAP function can be called.
	bt_l2cap_is_channel_open	This is function bt_l2cap_is_channel_open.
	bt_l2cap_listen_ext	brief Listen for incoming connections. ingroup l2cap details This function tells the L2CAP manager to listen for incoming connections on a specific PSM. When a connection is established a callback function is called. param mgr The L2CAP manager. param psm The PSM on which the manager will listen and accept incoming connections. param callback The callback function that will be called when an incoming connection is established. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE when the function succeeds. li c FALSE otherwise. The... more
	bt_l2cap_listen_fixed_channel	This is function bt_l2cap_listen_fixed_channel.
	bt_l2cap_reject	brief Send reject command (used to reject unknown or invalid commands)
	bt_l2cap_send_config	This is function bt_l2cap_send_config.
	bt_l2cap_update_conn_parameters	This is function bt_l2cap_update_conn_parameters.

Macros

	Name	Description
	bt_l2cap_connect	This is macro bt_l2cap_connect.
	bt_l2cap_listen	This is macro bt_l2cap_listen.
	L2CAP_DEFAULT_ERTX	seconds
	L2CAP_DEFAULT_RTX	seconds
	L2CAP_FRAME_TYPE_I	This is macro L2CAP_FRAME_TYPE_I.
	L2CAP_FRAME_TYPE_S	This is macro L2CAP_FRAME_TYPE_S.

	L2CAP_HCI_CONNECT_PACKET_TYPE	This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.
	L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE	This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.
	L2CAP_HCI_CONNECT_ROLE_SWITCH	This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.
	L2CAP_HEADER_LEN	This is macro L2CAP_HEADER_LEN.
	L2CAP_IDLE_HCI_CONNECTION_TIMEOUT	This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.
	L2CAP_MAX_MANAGERS	This is macro L2CAP_MAX_MANAGERS.
	L2CAP_MAX_MTU	This is macro L2CAP_MAX_MTU.
	L2CAP_MAX_RTX	This is macro L2CAP_MAX_RTX.
	L2CAP_MGR_STATE_FREE	This is macro L2CAP_MGR_STATE_FREE.
	L2CAP_MGR_STATE_LISTENING_L2CAP	This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.
	L2CAP_MGR_STATE_USED	This is macro L2CAP_MGR_STATE_USED.
	L2CAP_MIN_MTU	This is macro L2CAP_MIN_MTU.
	L2CAP_MONITOR_TIMEOUT	This is macro L2CAP_MONITOR_TIMEOUT.
	L2CAP_RETR_TIMEOUT	This is macro L2CAP_RETR_TIMEOUT.
	L2CAP_SAR_SDU_CONTINUE	This is macro L2CAP_SAR_SDU_CONTINUE.
	L2CAP_SAR_SDU_START	This is macro L2CAP_SAR_SDU_START.
	L2CAP_SAR_SDU_STOP	This is macro L2CAP_SAR_SDU_STOP.
	L2CAP_SAR_UNSEGMENTED	This is macro L2CAP_SAR_UNSEGMENTED.
	L2CAP_SFUNCTION_REJ	This is macro L2CAP_SFUNCTION_REJ.
	L2CAP_SFUNCTION_RNR	This is macro L2CAP_SFUNCTION_RNR.
	L2CAP_SFUNCTION_RR	This is macro L2CAP_SFUNCTION_RR.
	L2CAP_SFUNCTION_SREJ	This is macro L2CAP_SFUNCTION_SREJ.
	PSM_ATT	This is macro PSM_ATT.
	PSM_AVCTP	This is macro PSM_AVCTP.
	PSM_AVCTP_Browsing	This is macro PSM_AVCTP_Browsing.
	PSM_AVDTP	This is macro PSM_AVDTP.
	PSM_BNEP	This is macro PSM_BNEP.
	PSM_HID_Control	This is macro PSM_HID_Control.
	PSM_HID_Interrupt	This is macro PSM_HID_Interrupt.
	PSM_RFCOMM	This is macro PSM_RFCOMM.
	PSM_SDP	This is macro PSM_SDP.

Structures

	Name	Description
	_bt_l2cap_connect_params_s	This is record _bt_l2cap_connect_params_s.
	_bt_l2cap_fixed_channel_s	This is type bt_l2cap_fixed_channel_t.
	_bt_l2cap_mgr_s	This is type bt_l2cap_mgr_t.
	_bt_l2cap_psm_s	This is type bt_l2cap_psm_t.
	bt_l2cap_fixed_channel_t	This is type bt_l2cap_fixed_channel_t.
	bt_l2cap_mgr_p	This is type bt_l2cap_mgr_p.
	bt_l2cap_mgr_t	This is type bt_l2cap_mgr_t.
	bt_l2cap_psm_t	This is type bt_l2cap_psm_t.

Types

	Name	Description
	bt_l2cap_connect_callback_fp	This is type bt_l2cap_connect_callback_fp.
	bt_l2cap_connect_params_t	This is type bt_l2cap_connect_params_t.
	bt_l2cap_listen_callback_fp	This is type bt_l2cap_listen_callback_fp.

Description

l2cap_command_queue.h

Functions

	Name	Description
	_bt_l2cap_notify_and_remove	This is function _bt_l2cap_notify_and_remove.
	_bt_l2cap_send_commands_from_queue	This is function _bt_l2cap_send_commands_from_queue.

Macros

	Name	Description
	L2CAP_MAX_CMD_QUEUE_LEN	This is macro L2CAP_MAX_CMD_QUEUE_LEN.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_config.h

Macros

	Name	Description
	__L2CAP_CONFIG_H	This is macro __L2CAP_CONFIG_H.
	L2CAP_ALLOCATE_BUFFERS_FUNCTION	This is macro L2CAP_ALLOCATE_BUFFERS_FUNCTION.
	L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	L2CAP_ALLOCATE_BUFFERS_VARS	This is macro L2CAP_ALLOCATE_BUFFERS_VARS.
	L2CAP_DECL_ERETR_FUNCTIONS	This is macro L2CAP_DECL_ERETR_FUNCTIONS.
	L2CAP_FIXED_CHANNELS_DECL	This is macro L2CAP_FIXED_CHANNELS_DECL.
	L2CAP_HCI_PACKET_TYPE	• brief L2CAP_HCI_PACKET_TYPE. • ingroup btconfig * • details Defines a set of packets that link manager is allowed to use when calling bt_l2cap_connect. • The default value is to enable all packet types. enable all packet types
	L2CAP_HCI_PAGE_SCAN_REPETITION_MODE	brief L2CAP_HCI_PAGE_SCAN_REPETITION_MODE. ingroup btconfig details Defines a default value of the page scan repetition mode when calling bt_l2cap_connect. Must be set to one of the following values: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2 The default value is HCI_PAGE_SCAN_REPETITION_MODE_R0.
	L2CAP_HCI_ROLE_SWITCH	brief L2CAP_HCI_ROLE_SWITCH. ingroup btconfig details Defines a default value of the role switch parameter when calling bt_l2cap_connect. Must be set to one of the following values: HCI_ROLE_SWITCH_ALLOW HCI_ROLE_SWITCH_DISALLOW The default value is to allow the role switch.
	L2CAP_IDLE_CONNECTION_TIMEOUT	seconds
	L2CAP_MAX_FIXED_CHANNELS	This is macro L2CAP_MAX_FIXED_CHANNELS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_config_handlers.h

Macros

	Name	Description
	__L2CAP_CONFIG_HANDLERS_H	This is macro __L2CAP_CONFIG_HANDLERS_H.
	PACK_CONFIG_REQUEST	This is macro PACK_CONFIG_REQUEST.
	PACK_CONFIG_RESPONSE	This is macro PACK_CONFIG_RESPONSE.
	PACK_CONN_REQUEST	This is macro PACK_CONN_REQUEST.
	PACK_CONN_RESPONSE	This is macro PACK_CONN_RESPONSE.
	PACK_DCONN_REQUEST	This is macro PACK_DCONN_REQUEST.
	PACK_DCONN_RESPONSE	This is macro PACK_DCONN_RESPONSE.
	PACK_INFO_REQUEST	This is macro PACK_INFO_REQUEST.
	PACK_INFO_RESPONSE	This is macro PACK_INFO_RESPONSE.
	PROCESS_CONFIG_REQ	This is macro PROCESS_CONFIG_REQ.
	PROCESS_CONFIG_RES	This is macro PROCESS_CONFIG_RES.
	PROCESS_CONN_REQ	This is macro PROCESS_CONN_REQ.
	PROCESS_CONN_RES	This is macro PROCESS_CONN_RES.
	PROCESS_DCONN_REQ	This is macro PROCESS_DCONN_REQ.
	PROCESS_DCONN_RES	This is macro PROCESS_DCONN_RES.
	PROCESS_INFO_REQ	This is macro PROCESS_INFO_REQ.
	PROCESS_INFO_RES	This is macro PROCESS_INFO_RES.
	READ_CONFIG_REQUEST	This is macro READ_CONFIG_REQUEST.
	READ_CONFIG_RESPONSE	This is macro READ_CONFIG_RESPONSE.
	READ_CONN_REQUEST	This is macro READ_CONN_REQUEST.
	READ_CONN_RESPONSE	This is macro READ_CONN_RESPONSE.
	READ_DCONN_REQUEST	This is macro READ_DCONN_REQUEST.
	READ_DCONN_RESPONSE	This is macro READ_DCONN_RESPONSE.
	READ_INFO_REQUEST	This is macro READ_INFO_REQUEST.
	READ_INFO_RESPONSE	This is macro READ_INFO_RESPONSE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_eretr.h

Enumerations

	Name	Description
	_bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.

Functions

	Name	Description
	_bt_l2cap_eretr_handle_xmit_event	This is function _bt_l2cap_eretr_handle_xmit_event.
	_bt_l2cap_eretr_recv	This is function _bt_l2cap_eretr_recv.
	_bt_l2cap_eretr_retr_frames	This is function _bt_l2cap_eretr_retr_frames.
	_bt_l2cap_eretr_send_data	This is function _bt_l2cap_eretr_send_data.
	_bt_l2cap_eretr_send_pending_frames	This is function _bt_l2cap_eretr_send_pending_frames.
	_bt_l2cap_eretr_send_smart_data	This is function _bt_l2cap_eretr_send_smart_data.
	_bt_l2cap_fcs	This is function _bt_l2cap_fcs.
	_bt_l2cap_recv_req_seq_and_fbit	This is function _bt_l2cap_recv_req_seq_and_fbit.

	_bt_l2cap_send_ack	This is function _bt_l2cap_send_ack.
	_bt_l2cap_send_i_or_rr_or_rnr	This is function _bt_l2cap_send_i_or_rr_or_rnr.
	_bt_l2cap_send_rej	This is function _bt_l2cap_send_rej.
	_bt_l2cap_send_rnr	This is function _bt_l2cap_send_rnr.
	_bt_l2cap_send_rr	This is function _bt_l2cap_send_rr.
	_bt_l2cap_send_rr_or_rnr	This is function _bt_l2cap_send_rr_or_rnr.

Macros

	Name	Description
	_bt_l2cap_start_monitor_timer	This is macro _bt_l2cap_start_monitor_timer.
	_bt_l2cap_start_monitor_timer_if_not_running	This is macro _bt_l2cap_start_monitor_timer_if_not_running.
	_bt_l2cap_start_retr_timer	This is macro _bt_l2cap_start_retr_timer.
	_bt_l2cap_start_retr_timer_if_not_running	This is macro _bt_l2cap_start_retr_timer_if_not_running.
	_bt_l2cap_stop_monitor_timer	This is macro _bt_l2cap_stop_monitor_timer.
	_bt_l2cap_stop_retr_timer	This is macro _bt_l2cap_stop_retr_timer.
	GET_F_BIT	This is macro GET_F_BIT.
	GET_FRAME_TYPE	This is macro GET_FRAME_TYPE.
	GET_P_BIT	This is macro GET_P_BIT.
	GET_REQ_SEQ	This is macro GET_REQ_SEQ.
	GET_S_FUNCTION	This is macro GET_S_FUNCTION.
	GET_SAR	This is macro GET_SAR.
	GET_TX_SEQ	This is macro GET_TX_SEQ.
	SET_F_BIT	This is macro SET_F_BIT.
	SET_FRAME_TYPE	This is macro SET_FRAME_TYPE.
	SET_P_BIT	This is macro SET_P_BIT.
	SET_REQ_SEQ	This is macro SET_REQ_SEQ.
	SET_S_FUNCTION	This is macro SET_S_FUNCTION.
	SET_SAR	This is macro SET_SAR.
	SET_TX_SEQ	This is macro SET_TX_SEQ.

Structures

	Name	Description
	_bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_fixed_channel.h

Functions

	Name	Description
	bt_l2cap_allocate_fixed_channel	This is function bt_l2cap_allocate_fixed_channel.
	bt_l2cap_find_fixed_channel	This is function bt_l2cap_find_fixed_channel.
	bt_l2cap_free_fixed_channel	This is function bt_l2cap_free_fixed_channel.

Macros

	Name	Description
	L2CAP_LINK_TYPE_BD_EDR	This is macro L2CAP_LINK_TYPE_BD_EDR.
	L2CAP_LINK_TYPE_LE	This is macro L2CAP_LINK_TYPE_LE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

l2cap_packet.h

Functions

	Name	Description
	bt_l2cap_packet_cmd_assembler	This is function bt_l2cap_packet_cmd_assembler.
	bt_l2cap_packet_data_assembler	This is function bt_l2cap_packet_data_assembler.

Macros

	Name	Description
	L2CAP_PACKET_DATA_TYPE_RAW	This is macro L2CAP_PACKET_DATA_TYPE_RAW.
	L2CAP_PACKET_DATA_TYPE_SMART	This is macro L2CAP_PACKET_DATA_TYPE_SMART.

Structures

	Name	Description
	_bt_l2cap_packet_t	This is type bt_l2cap_packet_t.
	bt_l2cap_packet_t	This is type bt_l2cap_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

l2cap_private.h

Functions

	Name	Description
	_bt_l2cap_clear_channel_cmd_queue	This is function _bt_l2cap_clear_channel_cmd_queue.
	_bt_l2cap_eretr_pack_config_request	This is function _bt_l2cap_eretr_pack_config_request.
	_bt_l2cap_process_connect_signal	This is function _bt_l2cap_process_connect_signal.
	_l2cap_allocate_buffers	Defined by OEM through library configuration
	_l2cap_data_receive_callback	From l2cap_recv.c
	_pack_cmd_reject	This is function _pack_cmd_reject.
	_pack_config_request	This is function _pack_config_request.
	_pack_config_response	This is function _pack_config_response.
	_pack_conn_param_update_request	This is function _pack_conn_param_update_request.
	_pack_conn_param_update_response	This is function _pack_conn_param_update_response.
	_pack_conn_request	This is function _pack_conn_request.
	_pack_conn_response	This is function _pack_conn_response.
	_pack_dconn_request	This is function _pack_dconn_request.
	_pack_dconn_response	This is function _pack_dconn_response.
	_pack_echo_request	This is function _pack_echo_request.
	_pack_echo_response	This is function _pack_echo_response.
	_pack_info_request	This is function _pack_info_request.
	_pack_info_response	This is function _pack_info_response.
	_process_config_req	From cmd_config.c
	_process_config_res	This is function _process_config_res.
	_process_conn_param_update_req	This is function _process_conn_param_update_req.
	_process_conn_param_update_res	This is function _process_conn_param_update_res.
	_process_conn_req	From cmd_connect.c
	_process_conn_res	This is function _process_conn_res.
	_process_dconn_req	From cmd_disconnect.c
	_process_dconn_res	This is function _process_dconn_res.

	_process_echo_req	From cmd_echo.c
	_process_echo_res	This is function _process_echo_res.
	_process_info_req	From cmd_info.c
	_process_info_res	This is function _process_info_res.
	_process_reject	From cmd_reject.c
	_process_unknown_req	From cmd_unknown.c
	_process_unknown_res	This is function _process_unknown_res.
	_read_cmd_reject	This is function _read_cmd_reject.
	_read_config_request	This is function _read_config_request.
	_read_config_response	This is function _read_config_response.
	_read_conn_param_update_request	This is function _read_conn_param_update_request.
	_read_conn_param_update_response	This is function _read_conn_param_update_response.
	_read_conn_request	This is function _read_conn_request.
	_read_conn_response	This is function _read_conn_response.
	_read_dconn_request	This is function _read_dconn_request.
	_read_dconn_response	This is function _read_dconn_response.
	_read_echo_request	This is function _read_echo_request.
	_read_echo_response	This is function _read_echo_response.
	_read_info_request	This is function _read_info_request.
	_read_info_response	This is function _read_info_response.
	read_command	From channel_cmd_recv.c

Macros

	Name	Description
	CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED	This is macro CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED.
	L2CAP_EXT_FEATURES_ENABLED	This is macro L2CAP_EXT_FEATURES_ENABLED.

Types

	Name	Description
	bt_l2cap_cmd_assembler_fp	This is type bt_l2cap_cmd_assembler_fp.
	bt_l2cap_cmd_parser_fp	This is type bt_l2cap_cmd_parser_fp.
	bt_l2cap_request_handler_fp	This is type bt_l2cap_request_handler_fp.
	bt_l2cap_response_handler_fp	This is type bt_l2cap_response_handler_fp.

Variables

	Name	Description
	_frame_buffer_headers	This is variable _frame_buffer_headers.
	_frame_buffers	This is variable _frame_buffers.
	_l2cap_channels	This is variable _l2cap_channels.
	_l2cap_cmd_assemblers	This is variable _l2cap_cmd_assemblers.
	_l2cap_cmd_buffer_headers	Global variables defined by OEM configuration
	_l2cap_cmd_buffers	This is variable _l2cap_cmd_buffers.
	_l2cap_cmd_frame_buffer	This is variable _l2cap_cmd_frame_buffer.
	_l2cap_cmd_frame_buffer_size	This is variable _l2cap_cmd_frame_buffer_size.
	_l2cap_cmd_parsers	This is variable _l2cap_cmd_parsers.
	_l2cap_connect_params	This is variable _l2cap_connect_params.
	_l2cap_connect_params_headers	This is variable _l2cap_connect_params_headers.
	_l2cap_eretr_handle_xmit_event_fp	This is variable _l2cap_eretr_handle_xmit_event_fp.
	_l2cap_eretr_pack_config_request_fp	This is variable _l2cap_eretr_pack_config_request_fp.
	_l2cap_eretr_recv_fp	This is variable _l2cap_eretr_recv_fp.
	_l2cap_eretr_send_data_fp	This is variable _l2cap_eretr_send_data_fp.
	_l2cap_eretr_send_smart_data_fp	This is variable _l2cap_eretr_send_smart_data_fp.
	_l2cap_fixed_channels	This is variable _l2cap_fixed_channels.
	_l2cap_hci_connect_packet_type	This is variable _l2cap_hci_connect_packet_type.
	_l2cap_hci_page_scan_repetition_mode	This is variable _l2cap_hci_page_scan_repetition_mode.

	_l2cap_hci_role_switch	This is variable _l2cap_hci_role_switch.
	_l2cap_idle_hci_connection_timeout	This is variable _l2cap_idle_hci_connection_timeout.
	_l2cap_max_channels	This is variable _l2cap_max_channels.
	_l2cap_max_cmd_buffers	This is variable _l2cap_max_cmd_buffers.
	_l2cap_max_fixed_channels	This is variable _l2cap_max_fixed_channels.
	_l2cap_max_psms	This is variable _l2cap_max_psms.
	_l2cap_psms	This is variable _l2cap_psms.
	_l2cap_request_handlers	This is variable _l2cap_request_handlers.
	_l2cap_response_handlers	This is variable _l2cap_response_handlers.
	_mgrs	In l2cap_mgr.c
	_ram_size_l2cap_buffers	This is variable _ram_size_l2cap_buffers.
	_ram_size_l2cap_mgr	This is variable _ram_size_l2cap_mgr.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_psm.h

Functions

	Name	Description
	bt_l2cap_allocate_psm	This is function bt_l2cap_allocate_psm.
	bt_l2cap_find_psm	This is function bt_l2cap_find_psm.
	bt_l2cap_free_psm	This is function bt_l2cap_free_psm.
	bt_l2cap_init_psms	This is function bt_l2cap_init_psms.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_signal.h

Functions

	Name	Description
	_bt_l2cap_init_signal	This is function _bt_l2cap_init_signal.
	_bt_l2cap_set_signal	This is function _bt_l2cap_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_test.h

Macros

	Name	Description
	bt_l2cap_test_enable_local_config	This is macro bt_l2cap_test_enable_local_config.
	bt_l2cap_test_enable_remote_config	This is macro bt_l2cap_test_enable_remote_config.

Variables

	Name	Description
	_enable_local_config	if set to FALSE, l2cap_send_config will do nothing
	_enable_remote_config	This is variable _enable_remote_config.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_timer.h

Functions

	Name	Description
	_bt_l2cap_get_tick_count	This is function _bt_l2cap_get_tick_count.
	_bt_l2cap_init_timer	This is function _bt_l2cap_init_timer.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

packet.h

Structures

	Name	Description
	_bt_packet_t	This is type bt_packet_t.
	bt_packet_t	This is type bt_packet_t.

Types

	Name	Description
	bt_packet_assembler_fp	This is type bt_packet_assembler_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

patch.h

Macros

	Name	Description
	_W	This is macro _W.
	DCRH	Number of configuration blocks
	H	PS key store (0 = use default)
	W	This is macro W.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

queue.h

Functions

	Name	Description
	bt_q_add	This is function bt_q_add.
	bt_q_contains	This is function bt_q_contains.
	bt_q_get	This is function bt_q_get.
	bt_q_get_head	This is function bt_q_get_head.
	bt_q_get_length	This is function bt_q_get_length.
	bt_q_get_next	This is function bt_q_get_next.
	bt_q_push	This is function bt_q_push.
	bt_q_remove	This is function bt_q_remove.
	bt_q_remove_by_idx	This is function bt_q_remove_by_idx.

Structures

	Name	Description
	_bt_queue_element_t	This is type bt_queue_element_t.
	bt_queue_element_t	This is type bt_queue_element_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm.h

Functions

	Name	Description
	bt_rfcomm_allocate_dlc	brief Allocate DLC. ingroup rfcomm details This function allocates a new DLC on the specified RFCOMM session. param session The RFCOMM session. param dlci DLCI of the new DLC. return li A pointer to the new DLC if the function succeeds. li c NULL otherwise.
	bt_rfcomm_allocate_session	brief Allocate RFCOMM session. ingroup rfcomm details This function allocates a new RFCOMM session. param l2cap_mgr The L2CAP manager on which the RFCOMM session is to be created. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li A pointer to the new RFCOMM session structure if the function succeeds. li c NULL otherwise.
	bt_rfcomm_cancel_listen	This is function bt_rfcomm_cancel_listen.
	bt_rfcomm_close_dlc	brief Close DLC. ingroup rfcomm details This function closes a DLC. If DLCI = 0, the parent RFCOMM session is also closed.
	bt_rfcomm_connect	brief Connect to a remote device. ingroup rfcomm details This function establishes an RFCOMM connection with a remote device and opens a data DLC. Changes in the session state are reported through a callback function specified when the session has been allocated via call to bt_rfcomm_allocate_session. Changes in the data DLC are reported through a callback function specified in this call. param remote_addr Address of the remote device. param server_channel A server channel of the remote RFCOMM server. param callback The callback function for reporting changes in DLC state opened by this call. param param An arbitrary data pointer... more
	bt_rfcomm_find_dlc	brief Find DLC
	bt_rfcomm_free_dlc	brief Release DLC. ingroup rfcomm details This function releases the specified DLC. param dlc The DLC to be released.

	bt_rfcomm_free_session	brief Release RFCOMM session. ingroup rfcomm details This function deallocates the specified RFCOMM session. This function does not disconnect the session. It just frees the memory used by the bt_rfcomm_session structure. The session has to be disconnected by calling bt_rfcomm_close_dlc with DLCI = 0 first. param session The RFCOMM session to be deallocated.
	bt_rfcomm_get_frame_length	This is function bt_rfcomm_get_frame_length.
	bt_rfcomm_init	brief Initialize the RFCOMM layer. ingroup rfcomm details This function initializes the RFCOMM layer of the stack. It must be called prior to any other RFCOMM function can be called.
	bt_rfcomm_listen	brief Listen for incoming connections. ingroup rfcomm details This function enables incoming connections on the specified RFCOMM session. Changes in the session state are reported through a callback function. param server_channel A server channel on which the RFCOMM session will listen and accept incoming connections. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_rfcomm_open_dlc	brief Open DLC. ingroup rfcomm details This function opens the specified DLC. Before calling this function the RFCOMM session must be already open. This function is not to be used with DLCI = 0. Changes in DLC state are reported through a callback function. param dlc The DLC to open. param callback The callback function for reporting changes in DLC state. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in... more
	bt_rfcomm_send_credit	This is function bt_rfcomm_send_credit.
	bt_rfcomm_send_data	brief Send data over a DLC. ingroup rfcomm details This function sends data over the specified DLC. Operation completion is reported through callback function. param dlc The DLC. param data A pointer to the data to be sent. param len Data length. param callback The callback function that is called when operation completes. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

Macros

	Name	Description
	MK_CMD_ADDRESS	define RFCOMM_MS_RTC
	MK_DLCI	This is macro MK_DLCI.
	RFCOMM_CFC_ENABLED	This is macro RFCOMM_CFC_ENABLED.
	RFCOMM_CFC_LOCAL_CREDIT	This is macro RFCOMM_CFC_LOCAL_CREDIT.
	RFCOMM_CFC_MAX_INITIAL_CREDIT	This is macro RFCOMM_CFC_MAX_INITIAL_CREDIT.
	RFCOMM_CMD_STATUS_FC_PENDING	This is macro RFCOMM_CMD_STATUS_FC_PENDING.
	RFCOMM_CMD_STATUS_PENDING	This is macro RFCOMM_CMD_STATUS_PENDING.
	RFCOMM_CMD_STATUS_WAITING_RESPONSE	This is macro RFCOMM_CMD_STATUS_WAITING_RESPONSE.
	RFCOMM_COMMAND	This is macro RFCOMM_COMMAND.
	RFCOMM_CTL_MSG_CLD	This is macro RFCOMM_CTL_MSG_CLD.
	RFCOMM_CTL_MSG_FCOFF	This is macro RFCOMM_CTL_MSG_FCOFF.
	RFCOMM_CTL_MSG_FCON	This is macro RFCOMM_CTL_MSG_FCON.
	RFCOMM_CTL_MSG_MSC	This is macro RFCOMM_CTL_MSG_MSC.
	RFCOMM_CTL_MSG_NSC	This is macro RFCOMM_CTL_MSG_NSC.
	RFCOMM_CTL_MSG_PN	This is macro RFCOMM_CTL_MSG_PN.
	RFCOMM_CTL_MSG_PSC	This is macro RFCOMM_CTL_MSG_PSC.
	RFCOMM_CTL_MSG_RLS	This is macro RFCOMM_CTL_MSG_RLS.
	RFCOMM_CTL_MSG_RPN	This is macro RFCOMM_CTL_MSG_RPN.
	RFCOMM_CTL_MSG_SNC	This is macro RFCOMM_CTL_MSG_SNC.
	RFCOMM_CTL_MSG_TEST	This is macro RFCOMM_CTL_MSG_TEST.
	RFCOMM_DLC_CHANGED_CONN_STATE	This is macro RFCOMM_DLC_CHANGED_CONN_STATE.
	RFCOMM_DLC_CHANGED_REMOTE_MSC	This is macro RFCOMM_DLC_CHANGED_REMOTE_MSC.
	RFCOMM_DLC_CONNECTION_FAILED	This is macro RFCOMM_DLC_CONNECTION_FAILED.

	RFCOMM_DLC_STATE_CLOSED	This is macro RFCOMM_DLC_STATE_CLOSED.
	RFCOMM_DLC_STATE_OPEN	This is macro RFCOMM_DLC_STATE_OPEN.
	RFCOMM_DLCI_CONTROL	This is macro RFCOMM_DLCI_CONTROL.
	RFCOMM_DLCI_FREE	This is macro RFCOMM_DLCI_FREE.
	RFCOMM_ERR_DM	This is macro RFCOMM_ERR_DM.
	RFCOMM_ERR_INTERRUPTED	This is macro RFCOMM_ERR_INTERRUPTED.
	RFCOMM_ERR_SUCCESS	This is macro RFCOMM_ERR_SUCCESS.
	RFCOMM_ERR_TIMEOUT	This is macro RFCOMM_ERR_TIMEOUT.
	RFCOMM_FC_TYPE_AGREGATE	This is macro RFCOMM_FC_TYPE_AGREGATE.
	RFCOMM_FC_TYPE_CREDIT	This is macro RFCOMM_FC_TYPE_CREDIT.
	RFCOMM_FLAG_CR	This is macro RFCOMM_FLAG_CR.
	RFCOMM_FLAG_EA	This is macro RFCOMM_FLAG_EA.
	RFCOMM_FLAG_PF	This is macro RFCOMM_FLAG_PF.
	RFCOMM_FRAME_HEADER_LEN	This is macro RFCOMM_FRAME_HEADER_LEN.
	RFCOMM_FRAME_TYPE_DISC	This is macro RFCOMM_FRAME_TYPE_DISC.
	RFCOMM_FRAME_TYPE_DM	This is macro RFCOMM_FRAME_TYPE_DM.
	RFCOMM_FRAME_TYPE_SABM	This is macro RFCOMM_FRAME_TYPE_SABM.
	RFCOMM_FRAME_TYPE_UA	This is macro RFCOMM_FRAME_TYPE_UA.
	RFCOMM_FRAME_TYPE_UI	This is macro RFCOMM_FRAME_TYPE_UI.
	RFCOMM_FRAME_TYPE_UIH	This is macro RFCOMM_FRAME_TYPE_UIH.
	RFCOMM_MAX_INFO_LEN	This is macro RFCOMM_MAX_INFO_LEN.
	RFCOMM_MIX_INFO_LEN	This is macro RFCOMM_MIX_INFO_LEN.
	RFCOMM_MX_MSG_MAX_DATA_LEN	This is macro RFCOMM_MX_MSG_MAX_DATA_LEN.
	RFCOMM_RESPONSE	This is macro RFCOMM_RESPONSE.
	RFCOMM_ROLE_INITIATOR	This is macro RFCOMM_ROLE_INITIATOR.
	RFCOMM_ROLE_RESPONDER	This is macro RFCOMM_ROLE_RESPONDER.
	RFCOMM_RPN_BAUD_RATE_1152	This is macro RFCOMM_RPN_BAUD_RATE_1152.
	RFCOMM_RPN_BAUD_RATE_192	This is macro RFCOMM_RPN_BAUD_RATE_192.
	RFCOMM_RPN_BAUD_RATE_2304	This is macro RFCOMM_RPN_BAUD_RATE_2304.
	RFCOMM_RPN_BAUD_RATE_24	This is macro RFCOMM_RPN_BAUD_RATE_24.
	RFCOMM_RPN_BAUD_RATE_384	This is macro RFCOMM_RPN_BAUD_RATE_384.
	RFCOMM_RPN_BAUD_RATE_48	This is macro RFCOMM_RPN_BAUD_RATE_48.
	RFCOMM_RPN_BAUD_RATE_576	This is macro RFCOMM_RPN_BAUD_RATE_576.
	RFCOMM_RPN_BAUD_RATE_72	This is macro RFCOMM_RPN_BAUD_RATE_72.
	RFCOMM_RPN_BAUD_RATE_96	default
	RFCOMM_RPN_DATA_BIT_5	This is macro RFCOMM_RPN_DATA_BIT_5.
	RFCOMM_RPN_DATA_BIT_6	This is macro RFCOMM_RPN_DATA_BIT_6.
	RFCOMM_RPN_DATA_BIT_7	This is macro RFCOMM_RPN_DATA_BIT_7.
	RFCOMM_RPN_DATA_BIT_8	default
	RFCOMM_RPN_FLC_N	This is macro RFCOMM_RPN_FLC_N.
	RFCOMM_RPN_FLC_RTC_INPUT	This is macro RFCOMM_RPN_FLC_RTC_INPUT.
	RFCOMM_RPN_FLC_RTC_OUTPUT	This is macro RFCOMM_RPN_FLC_RTC_OUTPUT.
	RFCOMM_RPN_FLC_RTR_INPUT	This is macro RFCOMM_RPN_FLC_RTR_INPUT.
	RFCOMM_RPN_FLC_RTR_OUTPUT	This is macro RFCOMM_RPN_FLC_RTR_OUTPUT.
	RFCOMM_RPN_FLC_XONOFF_INPUT	This is macro RFCOMM_RPN_FLC_XONOFF_INPUT.
	RFCOMM_RPN_FLC_XONOFF_OUTPUT	This is macro RFCOMM_RPN_FLC_XONOFF_OUTPUT.
	RFCOMM_RPN_PARITY_EVEN	This is macro RFCOMM_RPN_PARITY_EVEN.
	RFCOMM_RPN_PARITY_MARK	This is macro RFCOMM_RPN_PARITY_MARK.
	RFCOMM_RPN_PARITY_N	default
	RFCOMM_RPN_PARITY_ODD	This is macro RFCOMM_RPN_PARITY_ODD.
	RFCOMM_RPN_PARITY_SPACE	This is macro RFCOMM_RPN_PARITY_SPACE.
	RFCOMM_RPN_PARITY_Y	This is macro RFCOMM_RPN_PARITY_Y.
	RFCOMM_RPN_STOP_BIT_1	default
	RFCOMM_RPN_STOP_BIT_1_5	This is macro RFCOMM_RPN_STOP_BIT_1_5.
	RFCOMM_RPN_XOFF_DEFAULT	This is macro RFCOMM_RPN_XOFF_DEFAULT.

	RFCOMM_RPN_XON_DEFAULT	This is macro RFCOMM_RPN_XON_DEFAULT.
	RFCOMM_SERIAL_PORT_CH_1	This is macro RFCOMM_SERIAL_PORT_CH_1.
	RFCOMM_SERIAL_PORT_CH_10	This is macro RFCOMM_SERIAL_PORT_CH_10.
	RFCOMM_SERIAL_PORT_CH_11	This is macro RFCOMM_SERIAL_PORT_CH_11.
	RFCOMM_SERIAL_PORT_CH_12	This is macro RFCOMM_SERIAL_PORT_CH_12.
	RFCOMM_SERIAL_PORT_CH_13	This is macro RFCOMM_SERIAL_PORT_CH_13.
	RFCOMM_SERIAL_PORT_CH_14	This is macro RFCOMM_SERIAL_PORT_CH_14.
	RFCOMM_SERIAL_PORT_CH_15	This is macro RFCOMM_SERIAL_PORT_CH_15.
	RFCOMM_SERIAL_PORT_CH_16	This is macro RFCOMM_SERIAL_PORT_CH_16.
	RFCOMM_SERIAL_PORT_CH_17	This is macro RFCOMM_SERIAL_PORT_CH_17.
	RFCOMM_SERIAL_PORT_CH_18	This is macro RFCOMM_SERIAL_PORT_CH_18.
	RFCOMM_SERIAL_PORT_CH_19	This is macro RFCOMM_SERIAL_PORT_CH_19.
	RFCOMM_SERIAL_PORT_CH_2	This is macro RFCOMM_SERIAL_PORT_CH_2.
	RFCOMM_SERIAL_PORT_CH_20	This is macro RFCOMM_SERIAL_PORT_CH_20.
	RFCOMM_SERIAL_PORT_CH_21	This is macro RFCOMM_SERIAL_PORT_CH_21.
	RFCOMM_SERIAL_PORT_CH_22	This is macro RFCOMM_SERIAL_PORT_CH_22.
	RFCOMM_SERIAL_PORT_CH_23	This is macro RFCOMM_SERIAL_PORT_CH_23.
	RFCOMM_SERIAL_PORT_CH_24	This is macro RFCOMM_SERIAL_PORT_CH_24.
	RFCOMM_SERIAL_PORT_CH_25	This is macro RFCOMM_SERIAL_PORT_CH_25.
	RFCOMM_SERIAL_PORT_CH_26	This is macro RFCOMM_SERIAL_PORT_CH_26.
	RFCOMM_SERIAL_PORT_CH_27	This is macro RFCOMM_SERIAL_PORT_CH_27.
	RFCOMM_SERIAL_PORT_CH_28	This is macro RFCOMM_SERIAL_PORT_CH_28.
	RFCOMM_SERIAL_PORT_CH_29	This is macro RFCOMM_SERIAL_PORT_CH_29.
	RFCOMM_SERIAL_PORT_CH_3	This is macro RFCOMM_SERIAL_PORT_CH_3.
	RFCOMM_SERIAL_PORT_CH_30	This is macro RFCOMM_SERIAL_PORT_CH_30.
	RFCOMM_SERIAL_PORT_CH_4	This is macro RFCOMM_SERIAL_PORT_CH_4.
	RFCOMM_SERIAL_PORT_CH_5	This is macro RFCOMM_SERIAL_PORT_CH_5.
	RFCOMM_SERIAL_PORT_CH_6	This is macro RFCOMM_SERIAL_PORT_CH_6.
	RFCOMM_SERIAL_PORT_CH_7	This is macro RFCOMM_SERIAL_PORT_CH_7.
	RFCOMM_SERIAL_PORT_CH_8	This is macro RFCOMM_SERIAL_PORT_CH_8.
	RFCOMM_SERIAL_PORT_CH_9	This is macro RFCOMM_SERIAL_PORT_CH_9.
	RFCOMM_SESSION_CHANGED_AFC	This is macro RFCOMM_SESSION_CHANGED_AFC.
	RFCOMM_SESSION_CHANGED_CONN_STATE	This is macro RFCOMM_SESSION_CHANGED_CONN_STATE.
	RFCOMM_SESSION_STATE_CONNECTED	This is macro RFCOMM_SESSION_STATE_CONNECTED.
	RFCOMM_SESSION_STATE_DISCONNECTED	This is macro RFCOMM_SESSION_STATE_DISCONNECTED.
	RFCOMM_SESSION_STATE_FREE	This is macro RFCOMM_SESSION_STATE_FREE.
	RFCOMM_TIMEOUT	This is macro RFCOMM_TIMEOUT.

Structures

	Name	Description
	_bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	_bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	_bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	_bt_rfcomm_session_t	This is type bt_rfcomm_session_t.
	bt_rfcomm_command_p	This is type bt_rfcomm_command_p.
	bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	bt_rfcomm_ctl_msg_p	This is type bt_rfcomm_ctl_msg_p.
	bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	bt_rfcomm_dlc_p	This is type bt_rfcomm_dlc_p.
	bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	bt_rfcomm_session_p	This is type bt_rfcomm_session_p.
	bt_rfcomm_session_t	This is type bt_rfcomm_session_t.

Types

	Name	Description
	bt_rfcomm_cmd_callback_fp	This is type bt_rfcomm_cmd_callback_fp.
	bt_rfcomm_dlc_state_callback_fp	This is type bt_rfcomm_dlc_state_callback_fp.
	bt_rfcomm_read_data_callback_fp	This is type bt_rfcomm_read_data_callback_fp.
	bt_rfcomm_send_data_callback_fp	This is type bt_rfcomm_send_data_callback_fp.
	bt_rfcomm_state_callback_fp	This is type bt_rfcomm_state_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_cmd_queue.h

Functions

	Name	Description
	rfcomm_cq_ack_cmd	This is function rfcomm_cq_ack_cmd.
	rfcomm_cq_ack_mx_cmd	This is function rfcomm_cq_ack_mx_cmd.
	rfcomm_cq_find_failed_pn	This is function rfcomm_cq_find_failed_pn.
	rfcomm_send_commands_from_queue	This is function rfcomm_send_commands_from_queue.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_config.h

Macros

	Name	Description
	__RFCOMM_CONFIG_H	This is macro __RFCOMM_CONFIG_H.
	RFCOMM_ALLOCATE_BUFFERS_FUNCTION	This is macro RFCOMM_ALLOCATE_BUFFERS_FUNCTION.
	RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	RFCOMM_ALLOCATE_BUFFERS_VARS	This is macro RFCOMM_ALLOCATE_BUFFERS_VARS.
	RFCOMM_BUFFER_SIZE	This is macro RFCOMM_BUFFER_SIZE.
	RFCOMM_ENABLE_MULTIDEVICE_CHANNELS	brief Enable multi-device server channels. ingroup rfcomm_config details Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.
	RFCOMM_INFO_LEN	brief Maximum size of the data portion of a UIH frame. ingroup rfcomm_config details This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI_L2CAP_BUFFER_LEN - RFCOMM_FRAME_HEADER_LEN - L2CAP_HEADER_LEN.

	RFCOMM_LOCAL_CREDIT	brief The number of receive buffers. ingroup rfcomm_config details This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory... more
	RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL	This is macro RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL.
	RFCOMM_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup rfcomm_config details This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as rn #define RFCOMM_MAX_CMD_BUFFERS (RFCOMM_MAX_DLCS - 1) * 2
	RFCOMM_MAX_DLCS	brief Maximum number of DLCs ingroup rfcomm_config details This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.
	RFCOMM_MAX_SERVER_CHANNELS	brief Maximum number of Server channels ingroup rfcomm_config details This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM_MAX_DLCS - 1.
	RFCOMM_MAX_SESSIONS	brief Maximum number of remote devices a local device can be connected to ingroup rfcomm_config details This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed HCI_MAX_HCI_CONNECTIONS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_mgr.h

Functions

	Name	Description
	_bt_rfcomm_allocate_channel	This is function _bt_rfcomm_allocate_channel.
	_bt_rfcomm_find_channel	This is function _bt_rfcomm_find_channel.
	_bt_rfcomm_get_mgr	This is function _bt_rfcomm_get_mgr.
	_bt_rfcomm_mgr_l2cap_listen_callback	This is function _bt_rfcomm_mgr_l2cap_listen_callback.
	_bt_rfcomm_mgr_notify_listeners	This is function _bt_rfcomm_mgr_notify_listeners.
	_rfcomm_init_mgr	This is function _rfcomm_init_mgr.
	bt_rfcomm_register_listener	This is function bt_rfcomm_register_listener.
	bt_rfcomm_unregister_listener	This is function bt_rfcomm_unregister_listener.

Structures

	Name	Description
	_bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	_bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	_bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.
	bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_mx.h

Functions

	Name	Description
	_rfcomm_allocate_mx_cmd	This is function _rfcomm_allocate_mx_cmd.
	_rfcomm_mx_process_fc	This is function _rfcomm_mx_process_fc.
	_rfcomm_mx_process_pn	This is function _rfcomm_mx_process_pn.
	_rfcomm_mx_process_rpn	This is function _rfcomm_mx_process_rpn.
	_rfcomm_process_mx_fc_response	This is function _rfcomm_process_mx_fc_response.
	_rfcomm_process_mx_msc_response	This is function _rfcomm_process_mx_msc_response.
	_rfcomm_process_mx_pn_response	This is function _rfcomm_process_mx_pn_response.
	_rfcomm_process_mx_rls_response	This is function _rfcomm_process_mx_rls_response.
	_rfcomm_send_mx_fc_cmd	This is function _rfcomm_send_mx_fc_cmd.
	_rfcomm_send_mx_msc_response	This is function _rfcomm_send_mx_msc_response.
	_rfcomm_send_mx_nsc_response	This is function _rfcomm_send_mx_nsc_response.
	_rfcomm_send_mx_rls_cmd	This is function _rfcomm_send_mx_rls_cmd.
	_rfcomm_send_mx_rls_response	This is function _rfcomm_send_mx_rls_response.
	_rfcomm_send_mx_test_response	This is function _rfcomm_send_mx_test_response.
	rfcomm_send_mx_msc_cmd	This is function rfcomm_send_mx_msc_cmd.
	rfcomm_send_mx_pn_cmd	This is function rfcomm_send_mx_pn_cmd.

Macros

	Name	Description
	RFCOMM_LINE_STATUS_FRAMING	This is macro RFCOMM_LINE_STATUS_FRAMING.
	RFCOMM_LINE_STATUS_OVERRUN	This is macro RFCOMM_LINE_STATUS_OVERRUN.
	RFCOMM_LINE_STATUS_PARITY	This is macro RFCOMM_LINE_STATUS_PARITY.
	RFCOMM_MODEM_STATUS_DV	This is macro RFCOMM_MODEM_STATUS_DV.
	RFCOMM_MODEM_STATUS_FC	This is macro RFCOMM_MODEM_STATUS_FC.
	RFCOMM_MODEM_STATUS_IC	This is macro RFCOMM_MODEM_STATUS_IC.
	RFCOMM_MODEM_STATUS_RTC	This is macro RFCOMM_MODEM_STATUS_RTC.
	RFCOMM_MODEM_STATUS_RTR	This is macro RFCOMM_MODEM_STATUS_RTR.
	RFCOMM_MX_MSG_FCOFF	This is macro RFCOMM_MX_MSG_FCOFF.
	RFCOMM_MX_MSG_FCON	This is macro RFCOMM_MX_MSG_FCON.
	RFCOMM_MX_MSG_MSC	This is macro RFCOMM_MX_MSG_MSC.
	RFCOMM_MX_MSG_NSC	This is macro RFCOMM_MX_MSG_NSC.
	RFCOMM_MX_MSG_PN	This is macro RFCOMM_MX_MSG_PN.
	RFCOMM_MX_MSG_RLS	This is macro RFCOMM_MX_MSG_RLS.
	RFCOMM_MX_MSG_RPN	This is macro RFCOMM_MX_MSG_RPN.
	RFCOMM_MX_MSG_TEST	This is macro RFCOMM_MX_MSG_TEST.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_private.h

Functions

	Name	Description
	_bt_rfcomm_clear_queue	From command_queue.c
	_bt_rfcomm_mgr_allocate_session	This is function _bt_rfcomm_mgr_allocate_session.
	_calc_fcs	This is function _calc_fcs.
	_rfcomm_alloc_cmd_buffer	This is function _rfcomm_alloc_cmd_buffer.
	_rfcomm_allocate_buffers	Defined by OEM through library configuration
	_rfcomm_free_cmd_buffer	This is function _rfcomm_free_cmd_buffer.
	_rfcomm_init_cmd_buffers	From rfcomm_cmdbuffer.c
	_rfcomm_init_sessions	From rfcomm_session.c
	_rfcomm_l2cap_read_data_callback	From rfcomm_cmd_recv.c
	_rfcomm_l2cap_state_changed_callback	From rfcomm.c
	_rfcomm_process_cmd_frame_disc	From frame_disc.c
	_rfcomm_process_cmd_frame_dm	This is function _rfcomm_process_cmd_frame_dm.
	_rfcomm_process_cmd_frame_sabm	This is function _rfcomm_process_cmd_frame_sabm.
	_rfcomm_process_cmd_frame_ua	This is function _rfcomm_process_cmd_frame_ua.
	_rfcomm_process_cmd_frame_uih	From frame_uih.c
	_rfcomm_process_res_frame_disc	This is function _rfcomm_process_res_frame_disc.
	_rfcomm_process_res_frame_dm	This is function _rfcomm_process_res_frame_dm.
	_rfcomm_process_res_frame_sabm	This is function _rfcomm_process_res_frame_sabm.
	_rfcomm_process_res_frame_ua	This is function _rfcomm_process_res_frame_ua.
	_rfcomm_process_res_frame_uih	This is function _rfcomm_process_res_frame_uih.
	_rfcomm_send_command	From rfcomm_cmd_send.c
	_rfcomm_send_dm_response	From frame_dm.c
	_rfcomm_send_sabm_cmd	From frame_sabm.c
	_rfcomm_send_ua_response	From frame_ua.c
	check_fcs	From rfcomm_fcs.c
	rfcomm_find_session	This is function rfcomm_find_session.
	rfcomm_send_cmd	This is function rfcomm_send_cmd.

Variables

	Name	Description
	_ram_size_rfcomm_buffers	This is variable _ram_size_rfcomm_buffers.
	_rfcomm_channels	This is variable _rfcomm_channels.
	_rfcomm_cmd_buffer_headers	This is variable _rfcomm_cmd_buffer_headers.
	_rfcomm_cmd_buffers	This is variable _rfcomm_cmd_buffers.
	_rfcomm_data_buffer_headers	This is variable _rfcomm_data_buffer_headers.
	_rfcomm_data_buffers	This is variable _rfcomm_data_buffers.
	_rfcomm_dlcs	This is variable _rfcomm_dlcs.
	_rfcomm_enable_multidevice_channels	This is variable _rfcomm_enable_multidevice_channels.
	_rfcomm_info_len	This is variable _rfcomm_info_len.
	_rfcomm_local_credit	This is variable _rfcomm_local_credit.
	_rfcomm_local_credit_send_threshold	This is variable _rfcomm_local_credit_send_threshold.
	_rfcomm_max_channels	This is variable _rfcomm_max_channels.
	_rfcomm_max_cmd_buffers	This is variable _rfcomm_max_cmd_buffers.
	_rfcomm_max_data_buffers	This is variable _rfcomm_max_data_buffers.
	_rfcomm_max_dlcs	This is variable _rfcomm_max_dlcs.
	_rfcomm_max_sessions	This is variable _rfcomm_max_sessions.
	_rfcomm_pdu_size	This is variable _rfcomm_pdu_size.
	_rfcomm_sessions	Global variables defined by OEM configuration

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_signal.h

Functions

	Name	Description
	_bt_rfcomm_init_signal	This is function _bt_rfcomm_init_signal.
	_bt_rfcomm_set_signal	This is function _bt_rfcomm_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_timer.h

Functions

	Name	Description
	_rfcomm_get_tick_count	This is function _rfcomm_get_tick_count.
	_rfcomm_init_timer	This is function _rfcomm_init_timer.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sbc.h

Functions

	Name	Description
	sbc_decode	Decodes the encoded SBC frame.
	sbc_get_codesize	Gets the input block size.
	sbc_get_frame_duration	Gets the time one input/output block takes to play in microseconds.
	sbc_get_frame_length	Gets the output block size.
	sbc_get_state_size	Gets the SBC Decoder's state data structure.
	sbc_init	Initializes the SBC Decoder.

Macros

	Name	Description
	__SBC_H	This is macro __SBC_H.
	INP_BUF_SIZE	Bytes
	MAX_SBC_DEC_STATE_SIZE	Bytes
	OUT_BUF_SIZE	Bytes
	SBC_AM_LOUDNESS	Allocation method
	SBC_AM_SNR	This is macro SBC_AM_SNR.
	SBC_BE	This is macro SBC_BE.

	SBC_BLK_12	This is macro SBC_BLK_12.
	SBC_BLK_16	This is macro SBC_BLK_16.
	SBC_BLK_4	Blocks
	SBC_BLK_8	This is macro SBC_BLK_8.
	SBC_FREQ_16000	Sampling frequency
	SBC_FREQ_32000	This is macro SBC_FREQ_32000.
	SBC_FREQ_44100	This is macro SBC_FREQ_44100.
	SBC_FREQ_48000	This is macro SBC_FREQ_48000.
	SBC_LE	Data endianness
	SBC_MODE_DUAL_CHANNEL	This is macro SBC_MODE_DUAL_CHANNEL.
	SBC_MODE_JOINT_STEREO	This is macro SBC_MODE_JOINT_STEREO.
	SBC_MODE_MONO	Channel mode
	SBC_MODE_STEREO	This is macro SBC_MODE_STEREO.
	SBC_SB_4	Sub-bands
	SBC_SB_8	This is macro SBC_SB_8.
	ssize_t	

Structures

	Name	Description
	sbc_struct	

Types

	Name	Description
	sbc_t	This is type sbc_t.

Description

Bluetooth low-complexity, subband codec (SBC) library

Copyright (C) 2008-2010 Nokia Corporation Copyright (C) 2004-2010 Marcel Holtmann Copyright (C) 2004-2005 Henryk Ploetz Copyright (C) 2005-2006 Brad Midgley

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

sdp.h

Functions

	Name	Description
	bt_sdp_start	brief Start SDP server ingroup sdp details This function starts the SDP server. param l2cap_mgr The L2CAP manager on which the SDP server is to be started. param sdp_db A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros. return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	BEGIN_DE_SEQUENCE	• brief Begin a data element sequence • ingroup sdp * • details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures. • The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute(). • For example, to find a AVRCP Target the following code can be used: code const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL }; const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL_TARGET }; BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID) END_DE_SEQUENCE(avrcp_target_service_search) . . . void findAvrcpTarget(void) { INIT_DE_SEQUENCE(avrcp_target_service_search); bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL); } endcode * • param... more
	DE_BOOL	brief Declare a boolean data element ingroup sdp details This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_INT	brief Declare a 1-byte signed integer data element ingroup sdp details This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_STRING	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_STRING2	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. param len The length of the data element value.
	DE_UINT	brief Declare a 1-byte unsigned integer data element ingroup sdp details This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.

	DE_UINT16	brief Declare a 2-byte unsigned integer data element ingroup sdp details This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_URL	brief Declare a URL data element ingroup sdp details This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value which is a pointer to a string.
	DE_UUID128	brief Declare a 128-bit UUID data element ingroup sdp details This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. The value must be a name of a variable of type bt_uuid.
	DE_UUID16	brief Declare a 16-bit UUID data element ingroup sdp details This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_UUID32	brief Declare a 32-bit UUID data element ingroup sdp details This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	END_DE_SEQUENCE	brief End a data element sequence ingroup sdp details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures. param id The data element sequence identifier.
	INIT_DE_SEQUENCE	brief Initialize a data element sequence ingroup sdp details This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence. param id The data element sequence identifier.
	SDP_ATTRID_AdditionalProtocolDescriptorLists	This is macro SDP_ATTRID_AdditionalProtocolDescriptorLists.
	SDP_ATTRID_BluetoothProfileDescriptorList	This is macro SDP_ATTRID_BluetoothProfileDescriptorList.
	SDP_ATTRID_BrowseGroupList	This is macro SDP_ATTRID_BrowseGroupList.
	SDP_ATTRID_ClientExecutableURL	This is macro SDP_ATTRID_ClientExecutableURL.
	SDP_ATTRID_DIPrimaryRecord	This is macro SDP_ATTRID_DIPrimaryRecord.
	SDP_ATTRID_DIProductId	This is macro SDP_ATTRID_DIProductId.
	SDP_ATTRID_DISpecificationId	This is macro SDP_ATTRID_DISpecificationId.
	SDP_ATTRID_DIVendorId	This is macro SDP_ATTRID_DIVendorId.
	SDP_ATTRID_DIVendorIdSource	This is macro SDP_ATTRID_DIVendorIdSource.
	SDP_ATTRID_DIVersion	This is macro SDP_ATTRID_DIVersion.
	SDP_ATTRID_DocumentationURL	This is macro SDP_ATTRID_DocumentationURL.
	SDP_ATTRID_GAPRemoteAudioVolumeControl	This is macro SDP_ATTRID_GAPRemoteAudioVolumeControl.
	SDP_ATTRID_GroupID	This is macro SDP_ATTRID_GroupID.
	SDP_ATTRID_HCRP_1284ID	This is macro SDP_ATTRID_HCRP_1284ID.
	SDP_ATTRID_HCRP_DeviceLocation	This is macro SDP_ATTRID_HCRP_DeviceLocation.
	SDP_ATTRID_HCRP_DeviceName	This is macro SDP_ATTRID_HCRP_DeviceName.
	SDP_ATTRID_HCRP_FriendlyName	This is macro SDP_ATTRID_HCRP_FriendlyName.
	SDP_ATTRID_HDPDataExchangeSpecification	This is macro SDP_ATTRID_HDPDataExchangeSpecification.
	SDP_ATTRID_HDPMCAPSupportedProcedures	This is macro SDP_ATTRID_HDPMCAPSupportedProcedures.
	SDP_ATTRID_HDPSuportedFeatures	This is macro SDP_ATTRID_HDPSuportedFeatures.
	SDP_ATTRID_HFPAGNetwork	This is macro SDP_ATTRID_HFPAGNetwork.
	SDP_ATTRID_HFPSupportedFeatures	This is macro SDP_ATTRID_HFPSupportedFeatures.
	SDP_ATTRID_HIDBatteryPower	This is macro SDP_ATTRID_HIDBatteryPower.
	SDP_ATTRID_HIDBootDevice	This is macro SDP_ATTRID_HIDBootDevice.
	SDP_ATTRID_HIDCountryCode	This is macro SDP_ATTRID_HIDCountryCode.

	SDP_ATTRID_HIDDescriptorList	This is macro SDP_ATTRID_HIDDescriptorList.
	SDP_ATTRID_HIDDeviceReleaseNumber	This is macro SDP_ATTRID_HIDDeviceReleaseNumber.
	SDP_ATTRID_HIDDeviceSubclass	This is macro SDP_ATTRID_HIDDeviceSubclass.
	SDP_ATTRID_HIDLANGIDBaseList	This is macro SDP_ATTRID_HIDLANGIDBaseList.
	SDP_ATTRID_HIDNormallyConnectable	This is macro SDP_ATTRID_HIDNormallyConnectable.
	SDP_ATTRID_HIDParserVersion	This is macro SDP_ATTRID_HIDParserVersion.
	SDP_ATTRID_HIDProfileVersion	This is macro SDP_ATTRID_HIDProfileVersion.
	SDP_ATTRID_HIDReconnectInitiate	This is macro SDP_ATTRID_HIDReconnectInitiate.
	SDP_ATTRID_HIDRemoteWake	This is macro SDP_ATTRID_HIDRemoteWake.
	SDP_ATTRID_HIDSDPDisable	This is macro SDP_ATTRID_HIDSDPDisable.
	SDP_ATTRID_HIDSupervisionTimeout	This is macro SDP_ATTRID_HIDSupervisionTimeout.
	SDP_ATTRID_HIDVirtualCable	This is macro SDP_ATTRID_HIDVirtualCable.
	SDP_ATTRID_IconURL	This is macro SDP_ATTRID_IconURL.
	SDP_ATTRID_INVALID	This is macro SDP_ATTRID_INVALID.
	SDP_ATTRID_LanguageBaseAttributeIDList	This is macro SDP_ATTRID_LanguageBaseAttributeIDList.
	SDP_ATTRID_OFFSET_ProviderName	This is macro SDP_ATTRID_OFFSET_ProviderName.
	SDP_ATTRID_OFFSET_ServiceDescription	This is macro SDP_ATTRID_OFFSET_ServiceDescription.
	SDP_ATTRID_OFFSET_ServiceName	This is macro SDP_ATTRID_OFFSET_ServiceName.
	SDP_ATTRID_PrimaryLanguageBaseId	This is macro SDP_ATTRID_PrimaryLanguageBaseId.
	SDP_ATTRID_ProtocolDescriptorList	This is macro SDP_ATTRID_ProtocolDescriptorList.
	SDP_ATTRID_ServiceAvailability	This is macro SDP_ATTRID_ServiceAvailability.
	SDP_ATTRID_ServiceClassIDList	This is macro SDP_ATTRID_ServiceClassIDList.
	SDP_ATTRID_ServiceDatabaseState	This is macro SDP_ATTRID_ServiceDatabaseState.
	SDP_ATTRID_ServiceID	This is macro SDP_ATTRID_ServiceID.
	SDP_ATTRID_ServiceInfoTimeToLive	This is macro SDP_ATTRID_ServiceInfoTimeToLive.
	SDP_ATTRID_ServiceRecordHandle	This is macro SDP_ATTRID_ServiceRecordHandle.
	SDP_ATTRID_ServiceRecordState	This is macro SDP_ATTRID_ServiceRecordState.
	SDP_ATTRID_SupportedFeatures	This is macro SDP_ATTRID_SupportedFeatures.
	SDP_ATTRID_VersionNumberList	This is macro SDP_ATTRID_VersionNumberList.
	SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION	This is macro SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION.
	SDP_CLSID_AUDIO_SINK	This is macro SDP_CLSID_AUDIO_SINK.
	SDP_CLSID_AUDIO_SOURCE	This is macro SDP_CLSID_AUDIO_SOURCE.
	SDP_CLSID_AV_REMOTE_CONTROL	This is macro SDP_CLSID_AV_REMOTE_CONTROL.
	SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER	This is macro SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER.
	SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID	This is macro SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID.
	SDP_CLSID_AV_REMOTE_CONTROL_TARGET	This is macro SDP_CLSID_AV_REMOTE_CONTROL_TARGET.
	SDP_CLSID_AVCTP	This is macro SDP_CLSID_AVCTP.
	SDP_CLSID_AVDTP	This is macro SDP_CLSID_AVDTP.
	SDP_CLSID_BrowseGroupDescriptorServiceClassID	This is macro SDP_CLSID_BrowseGroupDescriptorServiceClassID.
	SDP_CLSID_DialupNetworking	This is macro SDP_CLSID_DialupNetworking.
	SDP_CLSID_GENERIC_AUDIO	This is macro SDP_CLSID_GENERIC_AUDIO.
	SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT	This is macro SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT.
	SDP_CLSID_HARD_COPY_CONTROL_CHANNEL	This is macro SDP_CLSID_HARD_COPY_CONTROL_CHANNEL.
	SDP_CLSID_HARD_COPY_DATA_CHANNEL	This is macro SDP_CLSID_HARD_COPY_DATA_CHANNEL.
	SDP_CLSID_HARD_COPY_NOTIFICATION	This is macro SDP_CLSID_HARD_COPY_NOTIFICATION.
	SDP_CLSID_HCR_PRINT	This is macro SDP_CLSID_HCR_PRINT.
	SDP_CLSID_HCR_SCAN	This is macro SDP_CLSID_HCR_SCAN.
	SDP_CLSID_HDP	This is macro SDP_CLSID_HDP.
	SDP_CLSID_HDP_SINK	This is macro SDP_CLSID_HDP_SINK.
	SDP_CLSID_HDP_SOURCE	This is macro SDP_CLSID_HDP_SOURCE.
	SDP_CLSID_HFP	This is macro SDP_CLSID_HFP.
	SDP_CLSID_HFP_AG	This is macro SDP_CLSID_HFP_AG.
	SDP_CLSID_HID	This is macro SDP_CLSID_HID.
	SDP_CLSID_HIDProtocol	This is macro SDP_CLSID_HIDProtocol.
	SDP_CLSID_HSP	This is macro SDP_CLSID_HSP.

	SDP_CLSID_HSP_AG	This is macro SDP_CLSID_HSP_AG.
	SDP_CLSID_HSP_HS	This is macro SDP_CLSID_HSP_HS.
	SDP_CLSID_L2CAP	This is macro SDP_CLSID_L2CAP.
	SDP_CLSID_MCAP_CONTROL	This is macro SDP_CLSID_MCAP_CONTROL.
	SDP_CLSID_MCAP_DATA	This is macro SDP_CLSID_MCAP_DATA.
	SDP_CLSID_OBEXFileTransfer	This is macro SDP_CLSID_OBEXFileTransfer.
	SDP_CLSID_OBEXObjectPush	This is macro SDP_CLSID_OBEXObjectPush.
	SDP_CLSID_PBAP_PCE	This is macro SDP_CLSID_PBAP_PCE.
	SDP_CLSID_PBAP_PSE	This is macro SDP_CLSID_PBAP_PSE.
	SDP_CLSID_PNPInformation	This is macro SDP_CLSID_PNPInformation.
	SDP_CLSID_PublicBrowseGroup	This is macro SDP_CLSID_PublicBrowseGroup.
	SDP_CLSID_RFCOMM	This is macro SDP_CLSID_RFCOMM.
	SDP_CLSID_SerialPort	This is macro SDP_CLSID_SerialPort.
	SDP_CLSID_ServiceDiscoveryServerServiceClassID	defgroup sdp SDP This module describe functions and data structures used to start the SDP server and perform SDP queries.
	SDP_DATA_TYPE_ALTERNATIVE	This is macro SDP_DATA_TYPE_ALTERNATIVE.
	SDP_DATA_TYPE_BOOL	This is macro SDP_DATA_TYPE_BOOL.
	SDP_DATA_TYPE_INT	This is macro SDP_DATA_TYPE_INT.
	SDP_DATA_TYPE_INT128	This is macro SDP_DATA_TYPE_INT128.
	SDP_DATA_TYPE_INT16	This is macro SDP_DATA_TYPE_INT16.
	SDP_DATA_TYPE_INT32	This is macro SDP_DATA_TYPE_INT32.
	SDP_DATA_TYPE_INT64	This is macro SDP_DATA_TYPE_INT64.
	SDP_DATA_TYPE_INT8	This is macro SDP_DATA_TYPE_INT8.
	SDP_DATA_TYPE_NIL	This is macro SDP_DATA_TYPE_NIL.
	SDP_DATA_TYPE_SEQUENCE	This is macro SDP_DATA_TYPE_SEQUENCE.
	SDP_DATA_TYPE_STRING	This is macro SDP_DATA_TYPE_STRING.
	SDP_DATA_TYPE_UINT	This is macro SDP_DATA_TYPE_UINT.
	SDP_DATA_TYPE_UINT128	This is macro SDP_DATA_TYPE_UINT128.
	SDP_DATA_TYPE_UINT16	This is macro SDP_DATA_TYPE_UINT16.
	SDP_DATA_TYPE_UINT32	This is macro SDP_DATA_TYPE_UINT32.
	SDP_DATA_TYPE_UINT64	This is macro SDP_DATA_TYPE_UINT64.
	SDP_DATA_TYPE_UINT8	This is macro SDP_DATA_TYPE_UINT8.
	SDP_DATA_TYPE_URL	This is macro SDP_DATA_TYPE_URL.
	SDP_DATA_TYPE_UUID	This is macro SDP_DATA_TYPE_UUID.
	SDP_DATA_TYPE_UUID128	This is macro SDP_DATA_TYPE_UUID128.
	SDP_DATA_TYPE_UUID16	This is macro SDP_DATA_TYPE_UUID16.
	SDP_DATA_TYPE_UUID32	This is macro SDP_DATA_TYPE_UUID32.
	SDP_ERROR_INSUFFICIENT_RESOURCE	This is macro SDP_ERROR_INSUFFICIENT_RESOURCE.
	SDP_ERROR_INVALID_CONTINUATION_STATE	This is macro SDP_ERROR_INVALID_CONTINUATION_STATE.
	SDP_ERROR_INVALID_PDU_SIZE	This is macro SDP_ERROR_INVALID_PDU_SIZE.
	SDP_ERROR_INVALID_REQUEST_SYNTAX	This is macro SDP_ERROR_INVALID_REQUEST_SYNTAX.
	SDP_ERROR_INVALID_SDP_VERSION	This is macro SDP_ERROR_INVALID_SDP_VERSION.
	SDP_ERROR_INVALID_SR_HANDLE	This is macro SDP_ERROR_INVALID_SR_HANDLE.
	SDP_ERROR_RESERVED	This is macro SDP_ERROR_RESERVED.
	SDP_ErrorResponse	This is macro SDP_ErrorResponse.
	SDP_FTP_SERVICE_ID	This is macro SDP_FTP_SERVICE_ID.
	SDP_HID_SERVICE_ID	This is macro SDP_HID_SERVICE_ID.
	SDP_HSP_AG_SERVICE_ID	This is macro SDP_HSP_AG_SERVICE_ID.
	SDP_HSP_HS_SERVICE_ID	This is macro SDP_HSP_HS_SERVICE_ID.
	SDP_MAX_ATTRIBUTE_PATTERN_LEN	This is macro SDP_MAX_ATTRIBUTE_PATTERN_LEN.
	SDP_MAX_DATA_ELEMENT_LEN	This is macro SDP_MAX_DATA_ELEMENT_LEN.
	SDP_MAX_DATA_ELEMENTS	This is macro SDP_MAX_DATA_ELEMENTS.
	SDP_MAX_SEARCH_PATTERN_LEN	This is macro SDP_MAX_SEARCH_PATTERN_LEN.

	SDP_MAX_TRANSACTIONS	typedef struct _sdp_db { bt_int count; bt_sdp_service_record_p *records; } bt_sdp_db, *bt_sdp_db_p;
	SDP_PDU_HEADER_LEN	This is macro SDP_PDU_HEADER_LEN.
	SDP_RFCOMM_SERVICE_ID	This is macro SDP_RFCOMM_SERVICE_ID.
	SDP_ServiceAttributeRequest	This is macro SDP_ServiceAttributeRequest.
	SDP_ServiceAttributeResponse	This is macro SDP_ServiceAttributeResponse.
	SDP_ServiceSearchAttributeRequest	This is macro SDP_ServiceSearchAttributeRequest.
	SDP_ServiceSearchAttributeResponse	This is macro SDP_ServiceSearchAttributeResponse.
	SDP_ServiceSearchRequest	This is macro SDP_ServiceSearchRequest.
	SDP_ServiceSearchResponse	This is macro SDP_ServiceSearchResponse.
	SDP_SR_HANDLE_HDP_SINK	This is macro SDP_SR_HANDLE_HDP_SINK.
	SDP_SR_HANDLE_HDP_SOURCE	This is macro SDP_SR_HANDLE_HDP_SOURCE.
	SDP_SR_HANDLE_HFP_HF	This is macro SDP_SR_HANDLE_HFP_HF.
	SDP_SR_HANDLE_HID	This is macro SDP_SR_HANDLE_HID.
	SDP_SR_HANDLE_HID_KEYBOARD	This is macro SDP_SR_HANDLE_HID_KEYBOARD.
	SDP_SR_HANDLE_HSP_HS	This is macro SDP_SR_HANDLE_HSP_HS.
	SDP_SR_HANDLE_OBEXFileTransfer	This is macro SDP_SR_HANDLE_OBEXFileTransfer.
	SDP_SR_HANDLE_OBEXObjectPush	This is macro SDP_SR_HANDLE_OBEXObjectPush.
	SDP_SR_HANDLE_PNPINFORMATION	This is macro SDP_SR_HANDLE_PNPINFORMATION.
	SDP_SR_HANDLE_RFCOMM	This is macro SDP_SR_HANDLE_RFCOMM.
	SDP_SR_HANDLE_SERVER	This is macro SDP_SR_HANDLE_SERVER.
	SDP_SR_HANDLE_TEST	for simulating service search using continuation state

Structures

	Name	Description
	_bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	_bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	_bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	_bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	_bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	_bt_sdp_transaction_t	This is type bt_sdp_transaction_t.
	bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	bt_sdp_serialization_state_p	This is type bt_sdp_serialization_state_p.
	bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	bt_sdp_service_transaction_p	This is type bt_sdp_service_transaction_p.
	bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	bt_sdp_transaction_t	This is type bt_sdp_transaction_t.

Types

	Name	Description
	bt_sdp_data_element_cp	This is type bt_sdp_data_element_cp.
	bt_sdp_data_element_p	This is type bt_sdp_data_element_p.
	bt_sdp_sequence_cp	This is type bt_sdp_sequence_cp.
	bt_sdp_sequence_p	This is type bt_sdp_sequence_p.
	bt_sr_handle_p	This is type bt_sr_handle_p.
	bt_sr_handle_t	This is type bt_sr_handle_t.

Description

sdp_client.h

Functions

	Name	Description
	bt_sdp_request_service_attribute	brief Search attributes ingroup sdp details This function retrieves attribute values from a service record. param channel The L2CAP channel used to communicate to the remote SDP server. param sr The service record handle specifies the service record from which attribute values are to be retrieved. param pattern The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called for the first time. To define a... more
	bt_sdp_request_service_search	brief Search service records ingroup sdp details This function locates service records on a remote SDP server that match the given service search pattern. param channel The L2CAP channel used to communicate to the remote SDP server. param pattern The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called. To define a data element sequence... more

Macros

	Name	Description
	SDP_CLIENT_EVT_CONNECTED	This is macro SDP_CLIENT_EVT_CONNECTED.
	SDP_CLIENT_EVT_CONNECTION_FAILED	This is macro SDP_CLIENT_EVT_CONNECTION_FAILED.
	SDP_CLIENT_EVT_DISCONNECTED	This is macro SDP_CLIENT_EVT_DISCONNECTED.
	SDP_CLIENT_EVT_NULL	This is macro SDP_CLIENT_EVT_NULL.

Structures

	Name	Description
	bt_sdp_client_evt_connected_t	This is record bt_sdp_client_evt_connected_t.
	bt_sdp_client_evt_disconnected_t	This is record bt_sdp_client_evt_disconnected_t.

Types

	Name	Description
	bt_sdp_client_callback_fp	This is type bt_sdp_client_callback_fp.
	bt_sdp_read_de_callback_ftp	This is type bt_sdp_read_de_callback_ftp.
	bt_sdp_service_attribute_callback_fp	This is type bt_sdp_service_attribute_callback_fp.
	bt_sdp_service_search_callback_fp	This is type bt_sdp_service_search_callback_fp.

Description

sdp_config.h

Macros

	Name	Description
	__SDP_CONFIG_H	This is macro __SDP_CONFIG_H.

	SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup sdp_config Configuration
		• ingroup sdp
		*
		• This module describes parameters used to configure SDP.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI and L2CAP must always be present // SDP is required only if stack is running in dual mode. This is the default mode. // To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol // must be defined: // #define BT_BLE_SINGLE_MODE
		// HCI configuration parameters #define... more

	SDP_ALLOCATE_BUFFERS_VARS	This is macro SDP_ALLOCATE_BUFFERS_VARS.
	SDP_MAX_ATTRIBUTE_RESULT_LEN	brief Maximum number of attributes to find ingroup sdp_config details This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.
	SDP_MAX_PDU_BUFFERS	brief Maximum number of SDP server PDU buffers. ingroup sdp_config details This parameter defines the maximum number of responses the SDP server can send at the same time.
	SDP_MAX_SEARCH_RESULT_LEN	brief Maximum number of service records to find. ingroup sdp_config details This parameter defines the maximum number of service records the SDP server will return to the client.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_packet.h

Functions

	Name	Description
	bt_sdp_packet_assembler	This is function bt_sdp_packet_assembler.

Structures

	Name	Description
	_bt_sdp_packet_t	This is type bt_sdp_packet_t.
	bt_sdp_packet_t	This is type bt_sdp_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

sdp_private.h

Functions

	Name	Description
	_bt_sdp_client_init	This is function _bt_sdp_client_init.
	_sdp_alloc_svc_tran_buffer	This is function _sdp_alloc_svc_tran_buffer.
	_sdp_alloc_tran_buffer	This is function _sdp_alloc_tran_buffer.
	_sdp_find_svc_transaction	This is function _sdp_find_svc_transaction.
	_sdp_find_transaction	This is function _sdp_find_transaction.
	_sdp_free_svc_tran_buffer	This is function _sdp_free_svc_tran_buffer.
	_sdp_free_tran_buffer	This is function _sdp_free_tran_buffer.

	_sdp_init_tran_buffers	From sdp_tran_buffer.c
	bt_sdp_read_attribute	From sdp_db_utils.c
	sdp_compare_uuid_de	This is function sdp_compare_uuid_de.
	sdp_find_attributes	This is function sdp_find_attributes.
	sdp_find_service_records2	From sdp_utils.c

Structures

	Name	Description
	_bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	_bt_sdp_server_data_element_t	Private types
	_bt_sdp_server_record_t	This is type bt_sdp_server_record_t.
	bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	bt_sdp_server_data_element_t	Private types
	bt_sdp_server_record_t	This is type bt_sdp_server_record_t.

Variables

	Name	Description
	_ram_size_sdp_buffers	This is variable _ram_size_sdp_buffers.
	_sdp_client_max_buffers	This is variable _sdp_client_max_buffers.
	_sdp_client_packet_buffer_headers	Global variables defined by OEM configuration
	_sdp_client_packet_buffers	This is variable _sdp_client_packet_buffers.
	_sdp_found_attr_list_buffers	This is variable _sdp_found_attr_list_buffers.
	_sdp_found_attr_lists_buffers	This is variable _sdp_found_attr_lists_buffers.
	_sdp_found_sr_lists_buffers	This is variable _sdp_found_sr_lists_buffers.
	_sdp_max_attribute_result_len	This is variable _sdp_max_attribute_result_len.
	_sdp_max_buffers	This is variable _sdp_max_buffers.
	_sdp_max_search_result_len	This is variable _sdp_max_search_result_len.
	_sdp_packet_buffer_headers	This is variable _sdp_packet_buffer_headers.
	_sdp_packet_buffers	This is variable _sdp_packet_buffers.
	_sdp_service_tran_buffer_headers	This is variable _sdp_service_tran_buffer_headers.
	_sdp_service_tran_buffer_mgr	This is variable _sdp_service_tran_buffer_mgr.
	_sdp_service_tran_buffers	This is variable _sdp_service_tran_buffers.
	_sdp_start_fp	This is variable _sdp_start_fp.
	_sdp_tran_buffer_headers2	This is variable _sdp_tran_buffer_headers2.
	_sdp_tran_buffer_mgr2	This is variable _sdp_tran_buffer_mgr2.
	_sdp_tran_buffers2	This is variable _sdp_tran_buffers2.
	sdp_db_main2	In sdp_server.c
	sdp_db_main2_len	This is variable sdp_db_main2_len.

Description

sdp_utils.h

Functions

	Name	Description
	_sdp_get_de_data_len	This is function _sdp_get_de_data_len.
	_sdp_get_de_hdr_len	This is function _sdp_get_de_hdr_len.
	_sdp_read_de_header	This is function _sdp_read_de_header.
	_sdp_write_data_element	This is function _sdp_write_data_element.
	bt_sdp_de_to_uuid	This is function bt_sdp_de_to_uuid.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp.h

Enumerations

	Name	Description
	_bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	_bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	_bt_spp_send_status_e	Send operation status.
	bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	bt_spp_send_status_e	Send operation status.

Functions

	Name	Description
	bt_spp_allocate	• Allocate a serial port. The returned serial port is initially in the ::SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp_listen(). The p callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the ::SPP_MAX_PORTS configuration parameter. @param l2cap_mgr L2CAP manager. @param callback Pointer to... more
	bt_spp_cancel_listen	• Stop listening for incoming connections. This function stops the port to accept incoming connections on a given server channel. @param port Serial port. @param channel The RFCOMM server channel to accept connections on. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_cancel_receive	• Cancel receive data. If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called. If there is no receive operation in progress calling this function has no effect. @param port Serial port.
	bt_spp_cancel_send	• Cancel send data. If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the ::SPP_SEND_STATUS_CANCELED status. If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the ::SPP_SEND_STATUS_CANCELED status. If there is no send operation in progess calling this function has no effect. @param port Serial port.
	bt_spp_clr_port_options	This is function bt_spp_clr_port_options.

	bt_spp_connect	• Connect to a remote device. This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the ::SPP_PORT_EVENT_CONNECTION_FAILED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param remote_addr Bluetooth address of the remote device. @param channel RFCOMM server channel on which the connection is to be established. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_deallocate	• Deallocate serial port. This function deallocates the specified port structure and other resources associated with it. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate(). @param port Serial port structure to deallocate. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_disconnect	• Disconnect from the remote device. This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event. If the port is already in the disconnected state the function does nothing and the callback is not called. @param port Serial port.
	bt_spp_find_server	This is function bt_spp_find_server.
	bt_spp_find_server_ex	This is function bt_spp_find_server_ex.
	bt_spp_get_frame_length	• Get frame length. This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length. @return RFCOMM frame length in bytes.
	bt_spp_get_hci_connection	• Get SPP port's ACL connection. This function returns a pinter to the structure that describes the ACL connection this port is on. @return Pointer to ACL connection description if the port is connected, NULL otherwise.
	bt_spp_get_local_modem_status	• Get local device's TS 07.10 control signals. This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return local device's TS 07.10 control signals.
	bt_spp_get_remote_address	brief Get the address of the remote device this device is connected to. ingroup spp param port Serial port. return li c A pointer to bt_bdaddr structure that contains the address of the remote device.
	bt_spp_get_remote_modem_status	• Get remote device's TS 07.10 control signals. This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return remote device's TS 07.10 control signals.
	bt_spp_init	• Initialize the SPP module. This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.

	bt_spp_listen	• Listen for incoming connections. This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use. When a remote device successfully establishes a connection on the specified port the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param channel The RFCOMM server channel on which to listen for connections. @return c TRUE... more
	bt_spp_receive	• Receive data. This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the... more
	bt_spp_send	• Send data. This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the ::SPP_PORT_EVENT_SEND_PROGRESS event. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress. The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). @param port Serial... more
	bt_spp_set_dtr	• Set local device's RS-232 DTR signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more
	bt_spp_set_local_modem_status	• Set local device's TS 07.10 control signals. Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a command to the remote device this function... more
	bt_spp_set_port_options	This is function bt_spp_set_port_options.

	bt_spp_set_rts	• Set local device's RS-232 RTS signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more

Macros

	Name	Description
	SPP_PORT_OPTION_ENCRYPTED	This is macro SPP_PORT_OPTION_ENCRYPTED.
	SPP_PORT_OPTION_MASTER	This is macro SPP_PORT_OPTION_MASTER.
	SPP_PORT_OPTION_SECURE	This is macro SPP_PORT_OPTION_SECURE.
	SPP_PORT_TYPE_INCOMING	This is macro SPP_PORT_TYPE_INCOMING.
	SPP_PORT_TYPE_OUTGOING	This is macro SPP_PORT_TYPE_OUTGOING.
	SPP_RS232_CTS	This is macro SPP_RS232_CTS.
	SPP_RS232_DCD	This is macro SPP_RS232_DCD.
	SPP_RS232_DSR	• @defgroup spp Serial Port Profile (SPP) The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile. Here are the steps for using this API: • Call the bt_spp_init() function. • Allocate a serial port structure with bt_spp_allocate(). One of the parameters to this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes. • To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state... more
	SPP_RS232_DTR	This is macro SPP_RS232_DTR.
	SPP_RS232_RI	This is macro SPP_RS232_RI.
	SPP_RS232_RTS	This is macro SPP_RS232_RTS.

Structures

	Name	Description
	_bt_spp_port_t	• Serial port structure. This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

Types

	Name	Description
	bt_spp_find_server_callback_fp	This is type bt_spp_find_server_callback_fp.
	bt_spp_port_t	This is type bt_spp_port_t.
	bt_spp_receive_callback_fp	• Serial port receive callback. This callback function is called when a receive operation initiated by bt_spp_receive() completes. @param port Serial port on which the receive operation completed. @param bytes_received Number of received bytes. @param param Callback parameter that was specified when bt_spp_allocate() was called.

	bt_spp_send_callback_fp	• Serial port send callback. This callback function is called when a send operation initiated by bt_spp_send() completes. @param port Serial port on which the send operation completed. @param bytes_sent Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send(); @param status Completion status. It is one of the values defined in the ::bt_spp_send_status_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.
	bt_spp_state_callback_fp	• Serial port state callback. This callback function is called whenever the state of a serial port is changed. @param port Serial port which state has changed. @param evt Event describing the nature of state change. It is one of the values defined in the ::bt_spp_port_event_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp_config.h

Macros

	Name	Description
	__SPP_CONFIG_H	This is macro __SPP_CONFIG_H.
	SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	SPP_ALLOCATE_BUFFERS_VARS	This is macro SPP_ALLOCATE_BUFFERS_VARS.
	SPP_DECLARE_FRAME_BUFFERS	This is macro SPP_DECLARE_FRAME_BUFFERS.
	SPP_DISABLE_BUFFERING	This is macro SPP_DISABLE_BUFFERING.
	SPP_FRAME_BUFFERS_RAM_SIZE	This is macro SPP_FRAME_BUFFERS_RAM_SIZE.
	SPP_FRAME_BUFFERS_SIZE	This is macro SPP_FRAME_BUFFERS_SIZE.
	SPP_MAX_PORTS	brief Maximum number of SPP ports. ingroup spp_config details This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is FALSE (default) this value should be equal to RFCOMM_MAX_SERVER_CHANNELS. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS * RFCOMM_MAX_SESSIONS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp_private.h

Functions

	Name	Description
	_bt_spp_client_init	This is function _bt_spp_client_init.
	_bt_spp_find_port	This is function _bt_spp_find_port.
	_bt_spp_handle_rx	This is function _bt_spp_handle_rx.
	_bt_spp_handle_tx	This is function _bt_spp_handle_tx.
	_bt_spp_rfcomm_read_data_callback	This is function _bt_spp_rfcomm_read_data_callback.

	_bt_spp_rfcomm_send_data_callback	This is function _bt_spp_rfcomm_send_data_callback.

Variables

	Name	Description
	_ram_size_spp_buffers	This is variable _ram_size_spp_buffers.
	_spp_connect_device_address	This is variable _spp_connect_device_address.
	_spp_disable_buffering	This is variable _spp_disable_buffering.
	_spp_frame_buffers	This is variable _spp_frame_buffers.
	_spp_frame_len	This is variable _spp_frame_len.
	_spp_max_ports	This is variable _spp_max_ports.
	_spp_ports	This is variable _spp_ports.
	_spp_remaining_connect_attemtps	This is variable _spp_remaining_connect_attemtps.

Description

ssp.h

Enumerations

	Name	Description
	_SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	_SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	_SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	_SSP_MODE	This is type SSP_MODE.
	_SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.
	SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	SSP_MODE	This is type SSP_MODE.
	SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.

Functions

	Name	Description
	bt_ssp_evt_handler	This is function bt_ssp_evt_handler.
	bt_ssp_init	This is function bt_ssp_init.
	bt_ssp_read_local_oob_data	This is function bt_ssp_read_local_oob_data.
	bt_ssp_send_keypress_notification	This is function bt_ssp_send_keypress_notification.
	bt_ssp_send_user_confirmation	This is function bt_ssp_send_user_confirmation.
	bt_ssp_send_user_passkey	This is function bt_ssp_send_user_passkey.
	bt_ssp_set_io_capabilities	This is function bt_ssp_set_io_capabilities.
	bt_ssp_set_mode	This is function bt_ssp_set_mode.
	bt_ssp_set_oob_data	This is function bt_ssp_set_oob_data.

Macros

	Name	Description
	OOB_DATA_HASH_LENGTH	Remote OOB Data Request Event
	OOB_DATA_RANDOMIZER_LENGTH	This is macro OOB_DATA_RANDOMIZER_LENGTH.
	SSP_MAX_MANAGERS	This is macro SSP_MAX_MANAGERS.

Structures

	Name	Description
	_bt_ssp_io_capability	This is type bt_ssp_io_capability.

	_bt_ssp_keypress_notification	Keypress Notification Event
	_bt_ssp_oob_data	This is type bt_ssp_oob_data.
	_bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	_bt_ssp_user_confirmation_request	User Confirmation Request Event
	_bt_ssp_user_passkey_notification	User Passkey Notification Event
	_bt_ssp_user_passkey_request	User Passkey Request Event
	bt_ssp_io_capability	This is type bt_ssp_io_capability.
	bt_ssp_keypress_notification	Keypress Notification Event
	bt_ssp_oob_data	This is type bt_ssp_oob_data.
	bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	bt_ssp_user_confirmation_request	User Confirmation Request Event
	bt_ssp_user_passkey_notification	User Passkey Notification Event
	bt_ssp_user_passkey_request	User Passkey Request Event

Types

	Name	Description
	bt_spp_read_local_oob_data_callback_fp	This is type bt_spp_read_local_oob_data_callback_fp.

Variables

	Name	Description
	_bt_ssp_evt_handler	This is variable _bt_ssp_evt_handler.
	_bt_ssp_init	This is variable _bt_ssp_init.

Description

ssp_event.h

Enumerations

	Name	Description
	_SSP_EVENT	This is type SSP_EVENT.
	SSP_EVENT	This is type SSP_EVENT.

Description

ssp_event_handler.h

Functions

	Name	Description
	ssp_evt_io_capability_request	This is function ssp_evt_io_capability_request.
	ssp_evt_io_capability_response	This is function ssp_evt_io_capability_response.
	ssp_evt_keypress_notification	This is function ssp_evt_keypress_notification.
	ssp_evt_oob_data_request	This is function ssp_evt_oob_data_request.
	ssp_evt_ssp_complete	This is function ssp_evt_ssp_complete.
	ssp_evt_user_confirmation_request	This is function ssp_evt_user_confirmation_request.
	ssp_evt_user_passkey_notification	This is function ssp_evt_user_passkey_notification.
	ssp_evt_user_passkey_request	This is function ssp_evt_user_passkey_request.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ti.h

Functions

	Name	Description
	btx_ti_a3dp_sink_close_stream	This is function btx_ti_a3dp_sink_close_stream.
	btx_ti_a3dp_sink_codec_config	This is function btx_ti_a3dp_sink_codec_config.
	btx_ti_a3dp_sink_open_stream	This is function btx_ti_a3dp_sink_open_stream.
	btx_ti_a3dp_sink_start_stream	This is function btx_ti_a3dp_sink_start_stream.
	btx_ti_a3dp_sink_stop_stream	This is function btx_ti_a3dp_sink_stop_stream.
	btx_ti_avpr_debug	This is function btx_ti_avpr_debug.
	btx_ti_avpr_enable	This is function btx_ti_avpr_enable.
	btx_ti_drpb_enable_rf_calibration	This is function btx_ti_drpb_enable_rf_calibration.
	btx_ti_drpb_set_power_vector	This is function btx_ti_drpb_set_power_vector.
	btx_ti_drpb_tester_con_tx	This is function btx_ti_drpb_tester_con_tx.
	btx_ti_enable_deep_sleep	This is function btx_ti_enable_deep_sleep.
	btx_ti_enable_fast_clock_crystal	This is function btx_ti_enable_fast_clock_crystal.
	btx_ti_enable_low_power_scan	This is function btx_ti_enable_low_power_scan.
	btx_ti_enable_low_power_scan_default	This is function btx_ti_enable_low_power_scan_default.
	btx_ti_exec_script	This is function btx_ti_exec_script.
	btx_ti_exec_script_oem	This is function btx_ti_exec_script_oem.
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36	CC2560 Scripts (Service pack 2.36)
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44	CC2560 Scripts (Service pack 2.44)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10	CC2564 Scripts (Service pack 2.10)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12	CC2564 Scripts (Service pack 2.12)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14	CC2564 Scripts (Service pack 2.14)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_7	CC2564 Scripts (Service pack 2.7)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short	CC2564 Scripts (Service pack 2.8)
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn.
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1	CC2564B Scripts (Service pack 0.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn.

	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2	CC2564B Scripts (Service pack 0.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0	CC2564B Scripts (Service pack 1.0)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1	CC2564B Scripts (Service pack 1.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2	CC2564B Scripts (Service pack 1.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3	CC2564 Scripts (Service pack 2.3)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4	CC2564 Scripts (Service pack 2.4)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn.
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3	CC2564 Scripts (Service pack 1.3)
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init	This is function btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init.
	btx_ti_init_ble_controller	This is function btx_ti_init_ble_controller.
	btx_ti_init_controller	This is function btx_ti_init_controller.
	btx_ti_le_enable	This is function btx_ti_le_enable.
	btx_ti_set_afh_mode	This is function btx_ti_set_afh_mode.
	btx_ti_set_uart_baud_rate	This is function btx_ti_set_uart_baud_rate.
	btx_ti_write_bdaddr	This is function btx_ti_write_bdaddr.
	btx_ti_write_codec_config	This is function btx_ti_write_codec_config.
	btx_ti_write_hardware_register	This is function btx_ti_write_hardware_register.

Macros

	Name	Description
	BTX_TI_A3DP_ROLE_SINK	This is macro BTX_TI_A3DP_ROLE_SINK.
	BTX_TI_A3DP_ROLE_SOURCE	This is macro BTX_TI_A3DP_ROLE_SOURCE.
	BTX_TI_AVPR_DISABLE	This is macro BTX_TI_AVPR_DISABLE.
	BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE.
	BTX_TI_AVPR_ENABLE	This is macro BTX_TI_AVPR_ENABLE.
	BTX_TI_AVPR_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_LOAD_A3DP_CODE.
	BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE	This is macro BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE.

	btx_ti_get_script__CC2560_ServicePack	Alias for the latest script
	btx_ti_get_script__CC2564_BLE_Init	This is macro btx_ti_get_script__CC2564_BLE_Init.
	btx_ti_get_script__CC2564_ServicePack	Aliases for latest scripts
	btx_ti_get_script__CC2564B_AVPR_Init	This is macro btx_ti_get_script__CC2564B_AVPR_Init.
	btx_ti_get_script__CC2564B_BLE_Init	This is macro btx_ti_get_script__CC2564B_BLE_Init.
	btx_ti_get_script__CC2564B_ServicePack	Aliases for latest scripts
	BTX_TI_MODULATION_SCHEME_8_DPSK	This is macro BTX_TI_MODULATION_SCHEME_8_DPSK.
	BTX_TI_MODULATION_SCHEME_CW	This is macro BTX_TI_MODULATION_SCHEME_CW.
	BTX_TI_MODULATION_SCHEME_GFSK	This is macro BTX_TI_MODULATION_SCHEME_GFSK.
	BTX_TI_MODULATION_SCHEME_P4_DPSK	This is macro BTX_TI_MODULATION_SCHEME_P4_DPSK.
	BTX_TI_MODULATION_TYPE_EDR2	This is macro BTX_TI_MODULATION_TYPE_EDR2.
	BTX_TI_MODULATION_TYPE_EDR3	This is macro BTX_TI_MODULATION_TYPE_EDR3.
	BTX_TI_MODULATION_TYPE_GFSK	This is macro BTX_TI_MODULATION_TYPE_GFSK.
	BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS	This is macro BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS.
	BTX_TI_SBC_ALLOCATION_METHOD_SNR	This is macro BTX_TI_SBC_ALLOCATION_METHOD_SNR.
	BTX_TI_SBC_BLOCKS_12	This is macro BTX_TI_SBC_BLOCKS_12.
	BTX_TI_SBC_BLOCKS_16	This is macro BTX_TI_SBC_BLOCKS_16.
	BTX_TI_SBC_BLOCKS_4	This is macro BTX_TI_SBC_BLOCKS_4.
	BTX_TI_SBC_BLOCKS_8	This is macro BTX_TI_SBC_BLOCKS_8.
	BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL.
	BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO.
	BTX_TI_SBC_CHANNEL_MODE_MONO	This is macro BTX_TI_SBC_CHANNEL_MODE_MONO.
	BTX_TI_SBC_CHANNEL_MODE_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_STEREO.
	BTX_TI_SBC_SAMPLE_FREQUENCY_16000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_16000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_32000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_32000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_44100	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_44100.
	BTX_TI_SBC_SAMPLE_FREQUENCY_48000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_48000.
	BTX_TI_TEST_PATTERN_ALL_0	This is macro BTX_TI_TEST_PATTERN_ALL_0.
	BTX_TI_TEST_PATTERN_ALL_1	This is macro BTX_TI_TEST_PATTERN_ALL_1.
	BTX_TI_TEST_PATTERN_F0F0	This is macro BTX_TI_TEST_PATTERN_F0F0.
	BTX_TI_TEST_PATTERN_FF00	This is macro BTX_TI_TEST_PATTERN_FF00.
	BTX_TI_TEST_PATTERN_PN15	This is macro BTX_TI_TEST_PATTERN_PN15.
	BTX_TI_TEST_PATTERN_PN9	This is macro BTX_TI_TEST_PATTERN_PN9.
	BTX_TI_TEST_PATTERN_USER	This is macro BTX_TI_TEST_PATTERN_USER.
	BTX_TI_TEST_PATTERN_Z0Z0	This is macro BTX_TI_TEST_PATTERN_Z0Z0.

Structures

	Name	Description
	_btx_ti_codec_config_s	This is type btx_ti_codec_config_t.
	_btx_ti_exec_script_buffer_t	This is record _btx_ti_exec_script_buffer_t.
	_btx_ti_exec_script_oem_buffer_t	This is record _btx_ti_exec_script_oem_buffer_t.
	_btx_ti_script_t	This is type btx_ti_script_t.
	btx_ti_codec_config_t	This is type btx_ti_codec_config_t.
	btx_ti_script_t	This is type btx_ti_script_t.

Types

	Name	Description
	btx_ti_completion_callback_fp	This is type btx_ti_completion_callback_fp.
	btx_ti_exec_script_buffer_t	This is type btx_ti_exec_script_buffer_t.
	btx_ti_exec_script_oem_buffer_t	This is type btx_ti_exec_script_oem_buffer_t.
	btx_ti_exec_script_oem_callback_fp	This is type btx_ti_exec_script_oem_callback_fp.

Description

ti_private.h

Variables

	Name	Description
	g_btx_ti_hci_callback	Data in btx_ti_shared_data.c
	g_btx_ti_hci_callback_param	This is variable g_btx_ti_hci_callback_param.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

types.h

Macros

	Name	Description
	BYTE_SIZE	This is macro BYTE_SIZE.

Types

	Name	Description
	bt_byte	This is type bt_byte.
	bt_int	This is type bt_int.
	bt_long	This is type bt_long.
	bt_uint	This is type bt_uint.
	bt_ulong	This is type bt_ulong.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC.

Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

vcard_parser.h

Functions

	Name	Description
	bt_vcard_parse_fragment	This is function bt_vcard_parse_fragment.
	bt_vcard_parser_reset	This is function bt_vcard_parser_reset.

Macros

	Name	Description
	VCARD_CALL_TYPE_DIALED	This is macro VCARD_CALL_TYPE_DIALED.
	VCARD_CALL_TYPE_MISSED	This is macro VCARD_CALL_TYPE_MISSED.
	VCARD_CALL_TYPE_RECEIVED	This is macro VCARD_CALL_TYPE_RECEIVED.
	VCARD_EVT_PROPERTY_PARAM	This is macro VCARD_EVT_PROPERTY_PARAM.
	VCARD_EVT_PROPERTY_STARTED	This is macro VCARD_EVT_PROPERTY_STARTED.
	VCARD_EVT_PROPERTY_VALUE	This is macro VCARD_EVT_PROPERTY_VALUE.
	VCARD_EVT_VCARD_ENDED	This is macro VCARD_EVT_VCARD_ENDED.
	VCARD_EVT_VCARD_STARTED	This is macro VCARD_EVT_VCARD_STARTED.
	VCARD_PARAM_CALL_TYPE	This is macro VCARD_PARAM_CALL_TYPE.
	VCARD_PARAM_CELL	This is macro VCARD_PARAM_CELL.
	VCARD_PARAM_DIALED	This is macro VCARD_PARAM_DIALED.
	VCARD_PARAM_ENCODING	This is macro VCARD_PARAM_ENCODING.
	VCARD_PARAM_HOME	This is macro VCARD_PARAM_HOME.
	VCARD_PARAM_LANGUAGE	This is macro VCARD_PARAM_LANGUAGE.
	VCARD_PARAM_MISSED	This is macro VCARD_PARAM_MISSED.

	VCARD_PARAM_RECEIVED	This is macro VCARD_PARAM_RECEIVED.
	VCARD_PARAM_TYPE	This is macro VCARD_PARAM_TYPE.
	VCARD_PARAM_VALUE	This is macro VCARD_PARAM_VALUE.
	VCARD_PARAM_WORK	This is macro VCARD_PARAM_WORK.
	VCARD_PARSER_STATE_READ_PARAM_NAME	This is macro VCARD_PARSER_STATE_READ_PARAM_NAME.
	VCARD_PARSER_STATE_READ_PARAM_VALUE	This is macro VCARD_PARSER_STATE_READ_PARAM_VALUE.
	VCARD_PARSER_STATE_READ_TYPE_NAME	This is macro VCARD_PARSER_STATE_READ_TYPE_NAME.
	VCARD_PARSER_STATE_READ_TYPE_VALUE	This is macro VCARD_PARSER_STATE_READ_TYPE_VALUE.
	VCARD_PARSER_STATE_SKIP_PARAM	This is macro VCARD_PARSER_STATE_SKIP_PARAM.
	VCARD_PARSER_STATE_SKIP_TYPE	This is macro VCARD_PARSER_STATE_SKIP_TYPE.
	VCARD_PROPERTY_ADR	Delivery Address
	VCARD_PROPERTY_AGENT	vCard of Person Representing
	VCARD_PROPERTY_BDAY	Birthday
	VCARD_PROPERTY_BEGIN	This is macro VCARD_PROPERTY_BEGIN.
	VCARD_PROPERTY_CATEGORIES	Categories
	VCARD_PROPERTY_CLASS	Class information
	VCARD_PROPERTY_EMAIL	Electronic Mail Address
	VCARD_PROPERTY_END	This is macro VCARD_PROPERTY_END.
	VCARD_PROPERTY_FN	Formatted Name
	VCARD_PROPERTY_GEO	Geographic Position
	VCARD_PROPERTY_KEY	Public Encryption Key
	VCARD_PROPERTY_LABEL	Delivery
	VCARD_PROPERTY_LOGO	Organization Logo
	VCARD_PROPERTY_MAILER	Electronic Mail
	VCARD_PROPERTY_N	Structured Presentation of Name
	VCARD_PROPERTY_NICKNAME	Nickname
	VCARD_PROPERTY_NOTE	Comments
	VCARD_PROPERTY_ORG	Name of Organization
	VCARD_PROPERTY_PHOTO	Associated Image or Photo
	VCARD_PROPERTY_PROID	Product ID
	VCARD_PROPERTY_REV	Revision
	VCARD_PROPERTY_ROLE	Role within the Organization
	VCARD_PROPERTY_SORT_STRING	String used for sorting operations
	VCARD_PROPERTY_SOUND	Pronunciation of Name
	VCARD_PROPERTY_TEL	Telephone Number
	VCARD_PROPERTY_TITLE	Job
	VCARD_PROPERTY_TZ	Time Zone
	VCARD_PROPERTY_UID	Unique ID
	VCARD_PROPERTY_URL	Uniform Resource Locator
	VCARD_PROPERTY_VERSION	vCard Version
	VCARD_PROPERTY_X_IRMC_CALL_DATETIME	Time stamp

Structures

	Name	Description
	_bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.
	_bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	_bt_vcard_parser_t	This is type bt_vcard_parser_t.
	bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.
	bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	bt_vcard_parser_t	This is type bt_vcard_parser_t.

Types

	Name	Description
	bt_vcard_parser_callback_fp	This is type bt_vcard_parser_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

xml_parser.h

Functions

	Name	Description
	bt_xml_parse_fragment	This is function bt_xml_parse_fragment.
	bt_xml_parser_reset	This is function bt_xml_parser_reset.

Macros

	Name	Description
	XML_EVT_ATTR_NAME	This is macro XML_EVT_ATTR_NAME.
	XML_EVT_ATTR_VALUE	This is macro XML_EVT_ATTR_VALUE.
	XML_EVT_TAG_ENDED	This is macro XML_EVT_TAG_ENDED.
	XML_EVT_TAG_STARTED	This is macro XML_EVT_TAG_STARTED.
	XML_PARSER_STATE_READ_ATTR_NAME	This is macro XML_PARSER_STATE_READ_ATTR_NAME.
	XML_PARSER_STATE_READ_ATTR_VALUE	This is macro XML_PARSER_STATE_READ_ATTR_VALUE.
	XML_PARSER_STATE_READ_TAG_NAME	This is macro XML_PARSER_STATE_READ_TAG_NAME.
	XML_PARSER_STATE_READ_TAG_VALUE	This is macro XML_PARSER_STATE_READ_TAG_VALUE.
	XML_PARSER_STATE_SKIP_ATTR	This is macro XML_PARSER_STATE_SKIP_ATTR.
	XML_PARSER_STATE_SKIP_COMMENTS	This is macro XML_PARSER_STATE_SKIP_COMMENTS.
	XML_PARSER_STATE_SKIP_PROC_INST	This is macro XML_PARSER_STATE_SKIP_PROC_INST.
	XML_PARSER_STATE_SKIP_TAG	This is macro XML_PARSER_STATE_SKIP_TAG.
	XML_PARSER_STATE_WAIT_ATTR_NAME	This is macro XML_PARSER_STATE_WAIT_ATTR_NAME.
	XML_PARSER_STATE_WAIT_ATTR_VALUE	This is macro XML_PARSER_STATE_WAIT_ATTR_VALUE.
	XML_PARSER_STATE_WAIT_TAG_CLOSE	This is macro XML_PARSER_STATE_WAIT_TAG_CLOSE.
	XML_PARSER_STATE_WAIT_TAG_NAME	This is macro XML_PARSER_STATE_WAIT_TAG_NAME.

Structures

	Name	Description
	_bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.
	_bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	_bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	_bt_xml_parser_t	This is type bt_xml_parser_t.
	bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.
	bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	bt_xml_parser_t	This is type bt_xml_parser_t.

Types

	Name	Description
	bt_xml_parser_callback_pf	This is type bt_xml_parser_callback_pf.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

Bootloader Library Help

This section describes the Bootloader Library that is available in MPLAB Harmony.

Introduction

This library provides a software Bootloader Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Bootloader Library can be used to upgrade firmware on a target device without the need for an external programmer or debugger.

A demonstration application, which can be downloaded into the target PIC32 device using the Bootloader is included, which provides a personal computer host application to communicate with the Bootloader firmware running inside the PIC32 device. This personal computer application is used to perform erase and programming operations.

Bootloader Theory

A Bootloader is a small application that starts the operation of the device. A Bootloader does not fully operate the device, but can perform various functions prior to starting the main application. Such functions can include:

- System checks
- Firmware upgrades
- Application integrity verification
- Starting the application

Using the Library

This topic describes the basic architecture of the Bootloader Library and provides information and examples on its use.

Description

Interface Header File: `bootloader.h`

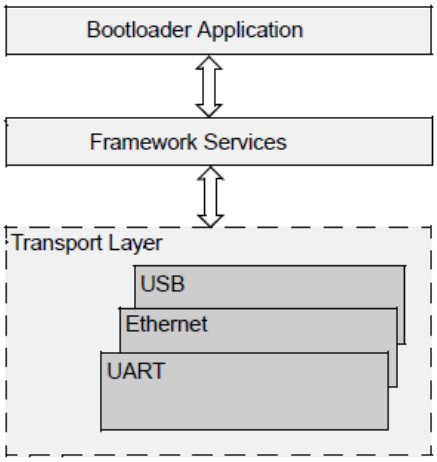
The interface to the Bootloader Library is defined in the `bootloader.h` header file. Any C language source (`.c`) file that uses the Bootloader Library should include `bootloader.h`.

Abstraction Model

This library provides the low-level abstraction of the Bootloader Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Bootloader Software Abstraction Block Diagram



Bootloader Application

The main purpose of the Bootloader Application layer is to implement the callback functions necessary to enhance the Bootloader. The various callback functions are described below, and are used to handle such tasks as forcing Bootloader mode, erasing and programming memory, and any cleanup before jumping to the application.

Framework Services

The Framework Services provide the main tasks of handling Flash memory. These include:

- Erasing Flash memory
- Programming hex file records into Flash memory
- Computing a CRC check of the Application in Program Memory
- Jumping to the Application

The Framework Services can be enhanced with the callback functions that can handle such things as external memory devices. Doing so requires that the hex files allocate the external memory assignments in a region that would not normally fall into the physical Flash region in the PIC32 device.

The Framework Services take an interface-agnostic approach to the actual communication medium. For the most part, it does not care whether the data comes from a PC Host via USB, UART, or Ethernet, or if it is coming by reading a file from a USB thumb drive or a SD Card. This allows the Bootloader to be expanded to handle other communication interfaces with minimal impact to the Framework Services.

Transport Layer

The Transport Layer, called Datastream in the Bootloader Framework code, is used to implement the specific interface to the source of the hex file.

Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Bootloader Library module.

How the Library Works

This topic provides information on how the library works.

Description

The Bootloader Library is implemented using a framework. The Bootloader firmware communicates with the personal computer host application by using a predefined communication protocol. The Bootloader Framework works in two different modes

Basic Mode

- Resides in the Boot Flash memory region
- The Bootloader performs program flash erase/program/verify operations while in the firmware upgrade mode
- Jumps to the application being programmed

Live-Update Mode(PIC32MZ Devices)

- Has the features of the Basic mode, but only programs the lower Flash region of the program Flash memory
- Also behaves as an application switcher to switch between the program flash panels when a LiveUpdate has occurred

Bootloader Flow

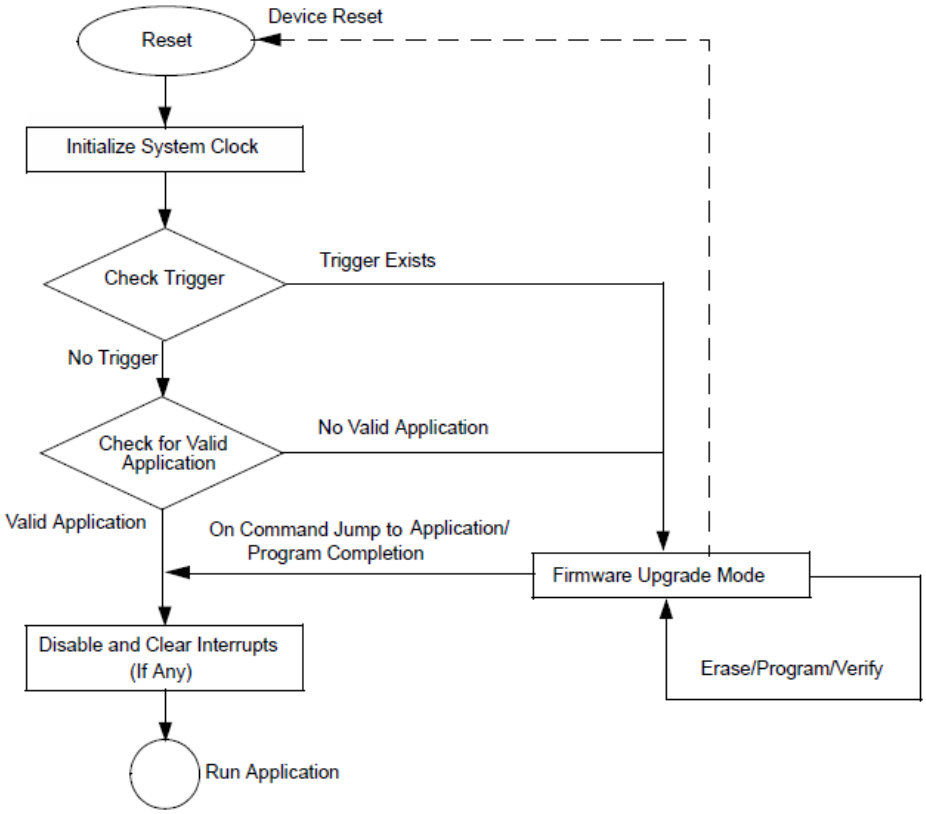
Describes the Basic and LiveUpdate flows of the PIC32 Bootloader .

Description

Basic Flow of the Bootloader

The Bootloader code starts executing on a device Reset. If there are no conditions to enter the firmware upgrade mode, the Bootloader starts executing the user application. The Bootloader performs Flash erase/program operations while in the firmware upgrade mode.

The following figure illustrates the Bootloader architecture. The Bootloader framework provides several API functions, which can be called by the Bootloaderapplication and the data stream layer. The Bootloader framework assists the user to easily modify the Bootloader application to adapt to different requirements. Refer to the Library Interface section for the available APIs.



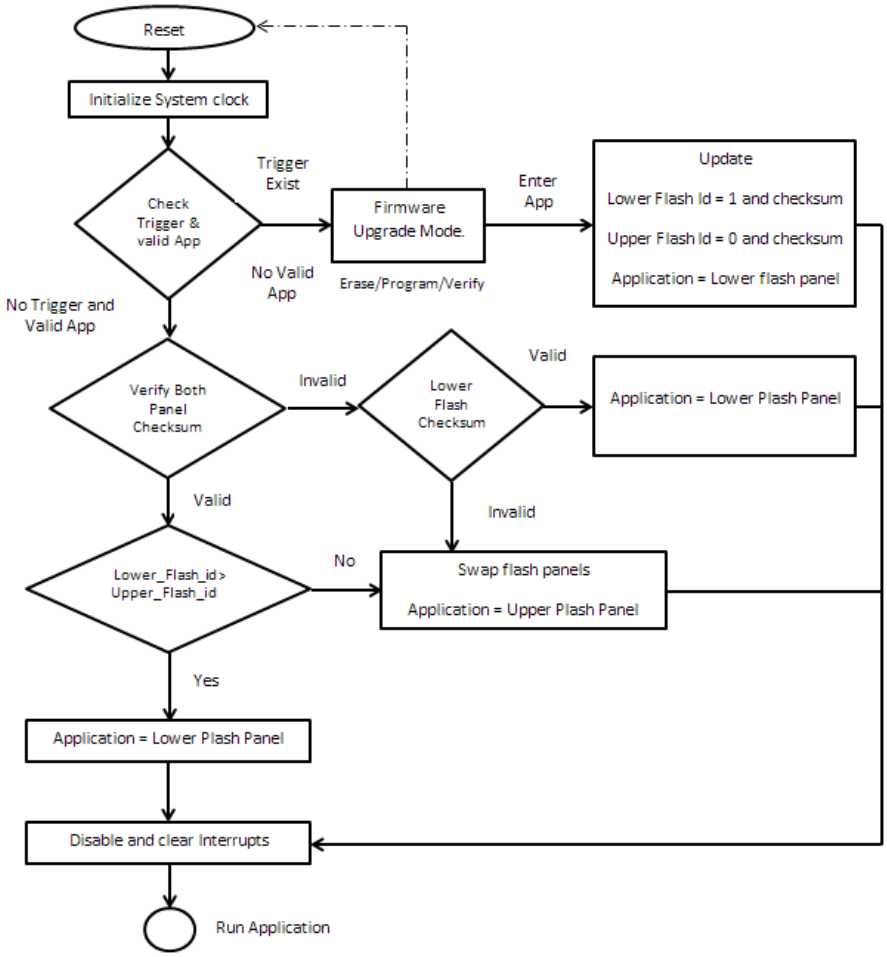
LiveUpdate Flow of the Bootloader

1. The Bootloader code starts executing on a device Reset.
2. If there are conditions to enter the firmware upgrade mode:

- The Bootloader performs erase/program operations while in the firmware upgrade mode for the lower Flash region of program Flash memory.
- Initializes the flash_id and flash_id_checksum of both program Flash panels
- Jumps to application in Panel 1

3. If there are no conditions to enter firmware upgrade mode:

- The Bootloader behaves as a Switcher
- It checks for the Flash ID being programmed in both panels (starting 16 bytes) by the application when the LiveUpdate occurred
- Switches to the panel with the higher Flash ID and jumps to the application programmed in that panel



Entering the Firmware Upgrade Mode

On a device Reset, the Bootloader forces itself into the firmware upgrade mode if the content of the user application's reset vector address is erased. During power-up press and hold switch SW1. While in firmware upgrade mode, the LED labeled LED1 will blink.

For your custom Bootloader, call the function, BOOTLOADER_ForceBootloadRegister, to register a custom function. From the callback function, the Bootloader can check whatever situation is necessary for forcing the Bootloader into Bootloader mode.

Exiting the Firmware Upgrade Mode

For USB HID, Ethernet, or the UART Bootloader, the firmware upgrade mode can be exited either by applying a hard Reset to the device, or by sending a "Jump to Application" command from the PC. For the USB Flash drive Bootloader, the firmware upgrade mode is exited either by a hard Reset or upon completion of firmware programming.

Note: The Bootloader should disable and clear any enabled interrupts before running the user application. The stray interrupts from the Bootloader may interfere with the user application and cause the application to fail. If applicable, interrupts and peripherals should be reinitialized in the user application.

Bootloader Placement in Memory

Describes the Bootloader placement in memory.

Description

Basic Memory Layout

The placement of the Bootloader in Flash memory controls several elements of the application and Bootloader interaction. Both have to be placed such that the application will not overwrite the Bootloader, and the Bootloader can properly program the application when it is downloaded.

PIC32M Layout

Figure 1 illustrates two schemes for the Bootloader placement based on the size of the Bootloader. Devices with a large enough Boot Flash

memory can place all of the Bootloader within Boot Flash. Fitting the Bootloader within the Boot Flash memory provides the complete program Flash memory for the user application.

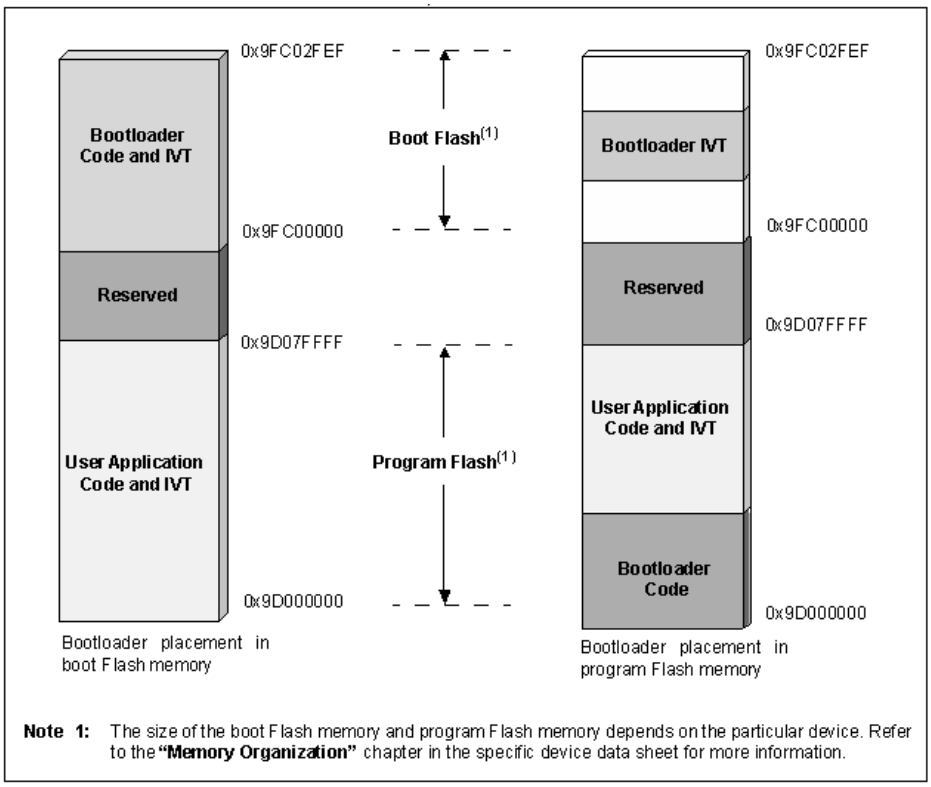
In the case of Bootloaders that exceed the size of PIC32 boot Flash, the Bootloader is split into two parts:

- For devices with 3 KB of Boot Flash, only the reset vector is placed in Boot Flash, with the Interrupt Vector Table (IVT) and C start-up code placed in Program Flash.
- For devices with 12 KB of Boot Flash, the Interrupt Vector Table (IVT) and the C start-up code are placed in Boot Flash, and the remaining portion of the Bootloader is placed inside the Program Flash.

The portion of the Bootloader that resides in Program Flash may be placed at either the beginning or the end of the Program Flash, depending on the needs of the application. The linker scripts generated by Harmony place the Bootloader code at the beginning of Program Flash (0x9D000000).

For PIC32MZ devices, with two 80 KB Boot Flash panels, the Bootloader may or may not fit entirely in one Boot Flash panel. In order to fit some of the Bootloaders, the linker script makes the two Boot Flash panels look like one contiguous Boot Flash memory. Unimplemented areas are blocked using a fill command to the linker.

Figure 1: Bootloader Memory Layout

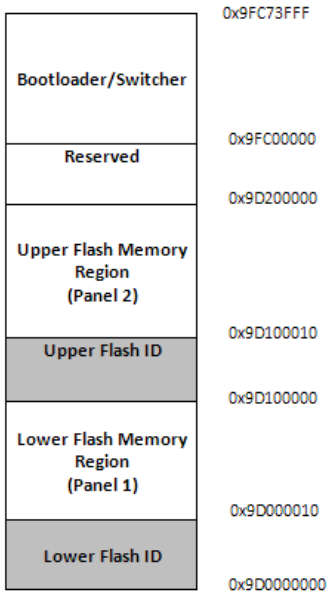


Note 1: The size of the boot Flash memory and program Flash memory depends on the particular device. Refer to the **"Memory Organization"** chapter in the specific device data sheet for more information.

LiveUpdate Memory Layout (PIC32MZ Devices)

The following figure shows the Flash ID structure.

```
typedef struct flash_id
{
   uint32_t checksum_start;
   uint32_t flash_id;
   uint32_t checksum_end;
   uint32_t dummy;
} T_FLASH_ID;
```



Handling Device Configuration Bits

Provides information on how device Configuration bits are handled on PIC32M devices.

Description

The Bootloader does not erase or write the device Configuration words while programming the new application firmware. This is because the device Configuration words settings are shared by both the Bootloader and the user application. Any modification to the device Configuration words may make the Bootloader non-functional. Therefore, it is highly recommended to have common device Configuration words settings for both the user application and the Bootloader.

When combining the Bootloader and Application Hex files in MPLAB X IDE, an error will be generated if the device Configuration words are different. This will be shown as a data conflict error, and the address given will match an address in the device Configuration words. This will indicate which word is in conflict, so that it can be resolved.

The Bootloader's Configuration words can be imported into the Application's MHC configuration file, thus simplifying the process of resolving differences.

Using the Bootloader Application (UART, USB HID, and Ethernet Bootloaders)

Provides information on using the Bootloader application.

Description

Refer to *Volume I: Getting Started With MPLAB Harmony > Applications Help > Bootloader Demonstrations > Demonstrations > basic > Running the Demonstration* for information on running the demonstration application.

Bootloader Communication Protocol (UART, USB HID, and Ethernet)

Provides information on the Bootloader communication protocol.

Description

The PC host application uses a communication protocol to interact with the Bootloader firmware. The PC host application acts as a master and issues commands to the Bootloader firmware to perform specific operations.

Frame Format

The communication protocol follows the frame format, as shown in Example 1. The frame format remains the same in both directions, that is, from the host application to the Bootloader, and from the Bootloader to the host application.

Example 1: Frame Format

```
[<SOH>…]<SOH>[<DATA>…]<CRCL><CRCH><EOT>
Where:
 <...> Represents a byte
 [...] Represents an optional or variable number of bytes
```

The frame starts with a control character, Start of Header (SOH), and ends with another control character, End of Transmission (EOT). The

integrity of the frame is protected by two bytes of Cyclic Redundancy Check (CRC)-16, represented by CRCL (low-byte) and CRCH (high-byte).

Control Characters

Some bytes in the Data field may imitate the control characters, SOH and EOT. The Data Link Escape (DLE) character is used to escape such bytes that could be interpreted as control characters. The Bootloader always accepts the byte following a <DLE> as data, and always sends a <DLE> before any of the control characters.

Table 1: Control Character Descriptions

Control	Hex Value	Description
<SOH>	0x01	Marks the beginning of a frame.
<EOT>	0x04	Marks the end of a frame.
<DLE>	0x10	Data link escape.

Commands

The PC host application can issue the commands listed in Table 2 to the Bootloader. The first byte in the data field carries the command.

Command Value in Hexadecimal	Description
0x01	Read the Bootloader version information.
0x02	Erase the Flash.
0x03	Program the Flash.
0x04	Read the CRC.
0x05	Jump to the application.

Read Bootloader Version Information

The PC host application request for version information to the Bootloader is shown in Example 2.

Example 2: Request

```
[<SOH>…]<SOH>[<0x01>]<CRCL><CRCH><EOT>
```

The Bootloader responds to the PC request for version information in two bytes, as shown in Example 3.

Example 3: Response

```
[<SOH>…]<SOH><0x01><MAJOR_VER><MINOR_VER>
<CRCL><CRCH><EOT>

Where:
MAJOR_VER = Major version of the Bootloader
MINOR_VER = Minor version of the Bootloader
```

Erase Flash

On receiving the erase Flash command from the PC host application, the Bootloader erases that part of the program Flash, which is allocated for the user application. The request frame from the PC host application to the Bootloader is shown in Example 4.

Example 4: Request

```
[<SOH>…]<SOH><0x02><CRCL><CRCH><EOT>
```

The response frame from the Bootloader to the PC host application is shown in Example 5.

Example 5: Response

```
[<SOH>…]<SOH><0x02><CRCL><CRCH><EOT>
```

Program Flash

The PC host application sends one or multiple hex records in Intel Hex format along with the program Flash command. The MPLAB XC32 C/C++ Compiler generates the image in the Intel Hex format. Each line in the Intel hexadecimal file represents a hexadecimal record. Each hexadecimal record starts with a colon (:) and is in ASCII format. The PC host application discards the colon and converts the remaining data from ASCII to hexadecimal, and then sends the data to the Bootloader. The Bootloader extracts the destination address and data from the hex record, and writes the data into program Flash.

The request frame from the PC host application to the Bootloader is shown in Example 6.

Example 6: Request

```
[<SOH>…]<SOH><0x03>[<HEX_RECORD>…]<CRCL>
<CRCH><EOT>
```

Where, `HEX_RECORD` is the Intel Hex record in hexadecimal format.

The response from the Bootloader to the PC host application is shown in Example 7.

Example 7: Response

```
[<SOH>…]<SOH><0x03><CRCL><CRCH><EOT>
```

Read CRC

The read CRC command is used to verify the content of the program Flash after programming. The request frame from the PC host application to the Bootloader is shown in Example 8.

Example 8: Request

```
[<SOH>…]<SOH><0x04><ADRS_LB><ADRS_HB>
<ADRS_UB><ADRS_MB><NUMBYTES_LB><NUMBYTES_HB>
<NUMBYTES_UB><NUMBYTES_MB><CRCL><CRCH><EOT>
```

ADRS_LB, ADRS_HB, ADRS_UB and ADRS_MB, as shown in Example 8, represent the 32-bit Flash addresses from where the CRC calculation begins.

NUMBYTES_LB, NUMBYTES_HB, NUMBYTES_UB and NUMBYTES_MB, as shown in Example 8, represent the total number of bytes in 32-bit format for which the CRC is to be calculated.

The response from the Bootloader to the PC host application is shown in Example 9.

Example 9: Response

```
[<SOH>…]<SOH><0x04><FLASH_CRCL><FLASH_CRCH>
<CRCL><CRCH><EOT>
```

Jump to Application

The Jump to Application command from the PC host application commands the Bootloader to execute the application. The request frame from the PC host application to the Bootloader is shown in Example 10.

Example 10: Request

```
[<SOH>…]<SOH><0x05><CRCL><CRCH><EOT>
```

There is no response to this command from the Bootloader because the Bootloader immediately exits the firmware upgrade mode and begins executing the application.

Demonstration Application

Provides information on the demonstration applications that can be used with the Bootloader.

Description

Refer to the following sections for information on the demonstrations that are available for the Bootloader:

- *Volume I: Getting Started With MPLAB Harmony > Applications Helps > Bootloader Demonstrations*
- *Volume I: Getting Started With MPLAB Harmony > Applications Help > Examples > Peripheral Library Examples > dma_led_pattern*

Generating the Application Linker Script

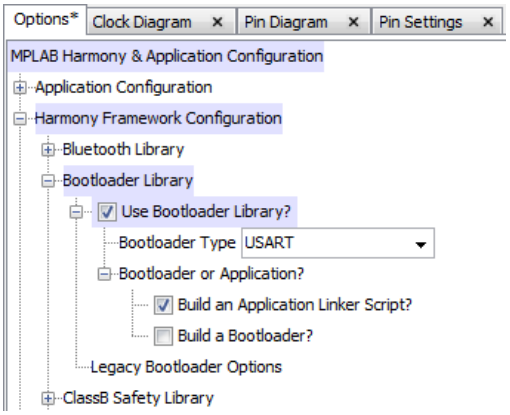
Provides information on generating the application linker script.

Description

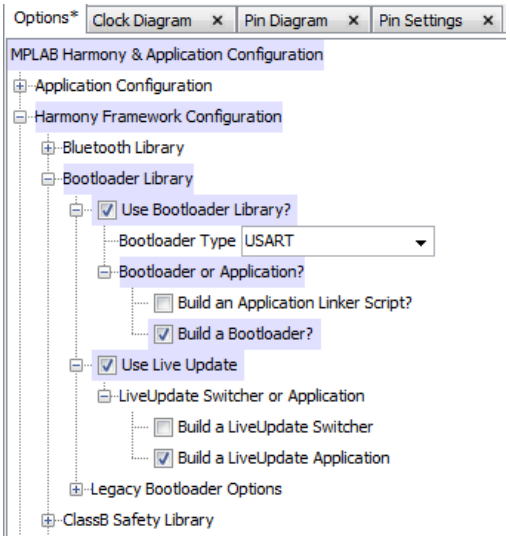
The MPLAB Harmony Configurator (MHC) can generate an application linker script specifically matched to the Bootloader type and mode in use. To add the linker script to your application, do the following:

1. In MPLAB X IDE, start the MHC, and open the configuration for your application.
2. Navigate to *Harmony Framework Configuration* > *Bootloader Library* and select **Use Bootloader Library?**, and other check boxes, as shown in the following figures.

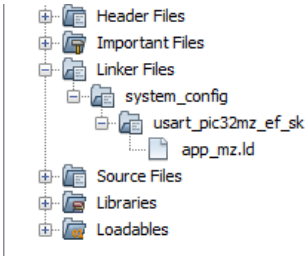
Basic Mode



LiveUpdate Mode (PIC32MZ Devices)



3. Select the Bootloader type in use using the Bootloader Type drop-down. This option must match between Bootloader and Application configurations. The linker scripts and reset address are configured for each based on the size of the Bootloader. Therefore, they must be the same for the Bootloader to look in the correct location for the application, and jump to the application's reset address.
4. When the application code is generated, a linker script will be created and inserted into the project folder, as shown in the following figure. The application linker scripts generated overrides the default linker script provided by the compiler.



Considerations While Moving the Application Image

Describes the procedure to place a user application into a desired program Flash memory region

Description

This section describes the procedure to place a user application into a desired program Flash memory region. It must be ensured that the user application's memory region does not overlap with the memory region reserved for the Bootloader.

The Bootloader generated by MHC should be considered a starting point for your product's Bootloader. As such, adding new features may cause the Bootloader to exceed the size intended in the Bootloader linker script. In addition, the Bootloaders provided in the `apps/bootloader/basic` folder are configured to the `-O1` compiler optimization. It is not the smallest possible size for the Bootloader, and turning on the `-Os` or MIPS16/microMIPS code options will reduce the size of the Bootloader.

If the size of the Bootloader changes, the following steps should be performed to adjust both the Bootloader and the application in order to make sure that both fit and make best use of the device memory.

1. Determine the new ending address in Program Flash for the Bootloader. This can be done by using either the `.map` file generated by MPLAB X IDE with the respective Compiler, or by using the ELFViewer plug-in for MPLAB X IDE.
2. Round the address up to the nearest page boundary. For devices with 1K page sizes, the address must end on a 0x400 boundary. For devices with 4K page sizes, the address must end on a 0x1000 boundary.
3. Within the Bootloader, change the following files:

- In `btx_mx.ld` for PIC32MX devices, change the length of kseg0_program_mem to the new boundary
- in `btl_cz.ld` for PIC32 C devices, change the Flash memory region LENGTH value to the new boundary

4. In system_config.h, change the value for APP_FLASH_BASE_ADDRESS to match the virtual address determined previously
5. Within the application, change the linker script, `app_mx.ld`, as follows:

- Change _ebase_address to the new boundary, plus enough pages to get the exceptions to start at an address on a 4K (0x1000) boundary. This is to align it on the boundary required by the MIPS core.
- Change _RESET_ADDR to the new boundary previously determined
- Change the ORIGIN value for kseg0_program_mem to the new boundary

6. Recompile both the Bootloader and the application, as well as test operations.

For PIC32MZ devices, any of the Bootloaders can fit entirely into Boot Flash, although some currently take advantage of both panels of Boot Flash. This is done by making the two Boot Flash memories appear as one panel in the linker script, but using a fill command to the linker to block out the area that is not available. When the resulting file is loaded by MPLAB X IDE, you may receive the following warning:

Warning: C:/microchip/harmony/core/apps/bootloader/basic/firmware/basic.X/dist/udp_pic32mz_ef_sk/production/basic.X.production.hex contains code that is located at addresses that do not exist on the PIC32MZxxx.

This warning can be ignored, as it is a warning about the fill memory.

Notes:

1. For more information on PIC32M linker script memory regions, refer to the *"MPLAB® XC32 Assembler, Linker and Utilities User's Guide"* (DS50002186) and the *"MPLAB® XC32 C/C++ Compiler User's Guide"* (DS50001686).

Bootloader Configurations

Provides information on the macros that are available to configure the Bootloader code during compilation.

Description

Most of the configuration options for the Bootloaders will be done with the driver or middleware-specific options available in MHC. For example:

- UART - Adjusting the port used and the baud rate
- USB - The value of the Vendor and Product IDs
- Ethernet - The default MAC IDs and IP Address to use
- SD Card - The SPI interface that communicates with the SD card

The following table lists macros that are modified in source code only, and their usage. Depending on user requirements, the values in these macros may need to be changed.

General Macros

Macro	Usage
APP_FLASH_BASE_ADDRESS	Base address of the program Flash reserved for the user application. The address value must point to the beginning of a Flash page, which is dependent upon the Flash page size of the specific device. Note that setting this value to an address below the maximum memory will allow your Bootloader to reserve Flash memory as needed.
APP_FLASH_END_ADDRESS	End address of the program Flash reserved for the user application. The address value must point to the end of a Flash page.
APP_RESET_ADDRESS	Address of user Reset vector. The Bootloader branches to this address when it must run the user application.
MAJOR_VERSION	Major version of the Bootloader firmware.
MINOR_VERSION	Minor version of the Bootloader firmware.

Configuring the Library

The configuration of the Bootloader Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Bootloader Library. Based on the selections made, the Bootloader Library will or will not support selected features. These configuration settings will apply to all instances of the Bootloader Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Bootloader Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/bootloader`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/src/bootloader.h	Includes all MPLAB Harmony-compatible function calls for the Bootloader Library.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`<install-dir>framework/bootloader/src/bootloader.c`	This file contains the core implementation of the Bootloader Library for PIC32M devices.
`<install-dir>framework/bootloader/src/datastream/datastream.c`	This file contains the data stream implementation of the Bootloader Library for PIC32M devices. In addition, a `datastream_<type>.c` file is included with this file. For example, `datastream_usart.c`.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`datastream_sdcard.c`	Data stream implementation when using the SD Card Bootloader configuration on PIC32M devices.
`datastream_udp.c`	Data stream implementation when using the Ethernet UDP Bootloader configuration on PIC32M devices.
`datastream_usart.c`	Data stream implementation when using the USART Bootloader configuration on PIC32M devices.
`datastream_usb_hid.c`	Data stream implementation when using the USB HID Device Bootloader configuration on PIC32M devices.
`datastream_usb_host.c`	Data stream implementation when using the USB Host Bootloader configuration on PIC32M devices.

Module Dependencies

The Bootloader Library will depend on the modules used for the communication medium, as listed in the following table.

Bootloader	Module Dependency and MHC Configuration
SD Card	SD Card Driver, configured to use the SPI port and Chip Select lines in use. File System, configured with the following MHC options: • Use File System Auto Mount Feature • Media Type: SYS_FS_MEDIA_TYPE_SD_CARD • FAT File System
USART	USART Static Driver
UDP	TCP/IP, configured with the following MHC options: • IPv4 • Use UDP • One Network Configuration, with interface, host name, and IP address as desired • Enable Stack Deinitialization Operations • Enable Configuration Save/Restore Functionality • Dynamic RAM Size: 20250
USB Device	USB, configured with the following MHC options: • USB Device • Number of Endpoints Used: 2 • Number of Functions: 1 • Device Class: HID • Vendor ID: 0x04d8 • Product ID: 0x003c • Manufacturer String: As desired • Product String: As desired

USB Host	USB, configured with the following MHC options: • USB Host (Recommended) • Use MSD Host Client Driver File System, configured with the following MHC options: • Use File System Auto Mount Feature • Media Type: SYS_FS_MEDIA_TYPE_MSD

Bootloader Sizing and Optimization

Provides Bootloader sizing and optimization information.

Description

The example Bootloaders provided in the `apps/bootloader/basic` folder have optimization settings for use by most MPLAB XC32 C/C++ Compiler users, which is `-O1`. However, in terms of size, this option does not produce the most optimal code. The following table lists the various Bootloaders, and their current size (based on Boot Flash size).

Flash Usage Based on Optimization in Bytes

Peripheral	Device Family	-O1	-Os	-Os MIPS16	-Os microAptiv
SD Card	PIC32MZ(1)	134968	122496	N/A	52110
UART	PIC32MX(1)	7376	6892	5696	N/A
UART	PIC32MZ(1)	12796	12228	N/A	10460
USB Device	PIC32MX	24944	22932	15328	N/A
USB Device	PIC32MZ(1)	38104	34976	N/A	26164
USB Host	PIC32MX	76156	66968	42808	N/A
USB Host	PIC32MZ(1)	85512	75944	N/A	53444
UDP	PIC32MX	73740	72416	48804	N/A
UDP	PIC32MZ(1)	90704	79856	N/A	Not Functional(2)

Notes:

1. The Bootloader resides entirely in Boot Flash. No Program Flash is in use.
2. The UDP Bootloader does not compile for PIC32MZ devices when microMIPS is selected.

Application Interrupt Vector Table Placement

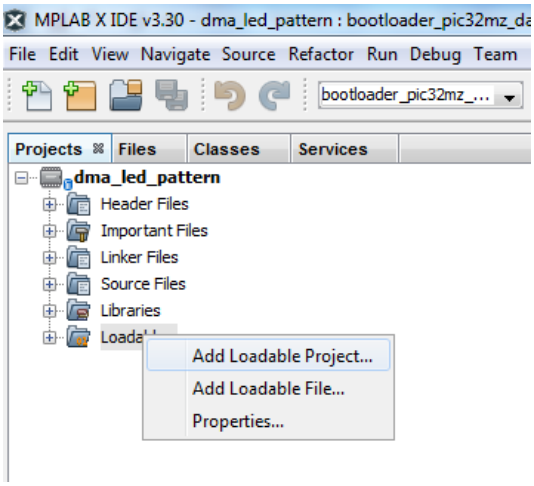
Beginning in MPLAB Harmony v1.08, the application linker scripts are able to have the interrupt vector table within the kseg0_program_mem, along with the rest of the application code. Generally, this means that it can be placed anywhere within the application code, and only the _ebase_address value in the application linker script needs to be updated.

Within the MIPS core, the base address of the interrupt vector table is set by the Ebase register (CP0 Register 15, Select 1). Because of the definition of this register, the interrupt vector table must be aligned on a 4K boundary. This applies to all PIC32 devices. The Bootloader will take the start of application Program Memory as the start of the application itself. This means that the interrupt vector table will need to be located at least at the next 4K boundary after that starting address.

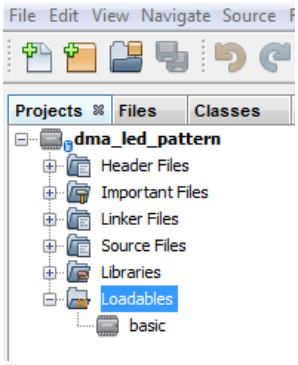
Debugging with a Bootloader and an Application

When developing a system that uses both an application and a Bootloader, it is useful to be able to run both as separate projects, but have them both in the device. This involves setting the Bootloader as a loadable project to the application. The steps to add the Bootloader as the loadable project, are as follows.

1. Right-click the application project's `Loadables` folder, and select **Add Loadable Project…**.



2. Navigate to the project folder of the Bootloader, selecting the correct configuration if there are multiple configurations, and click **Add**.



3. When the program is built, it will generate a single hex file that contains the combined Bootloader and application.
4. When debugging the program, both will be programmed into the PIC32. However, the Bootloader will be executed, and the PIC32 will halt when it reaches the main function of the application. To step through the Bootloader, it is necessary to set a breakpoint in the Bootloader code.

Commonly Encountered Situations

When creating the Bootloader and the application, you may encounter some of these situations while combining the two, and errors may be generated by the linker.

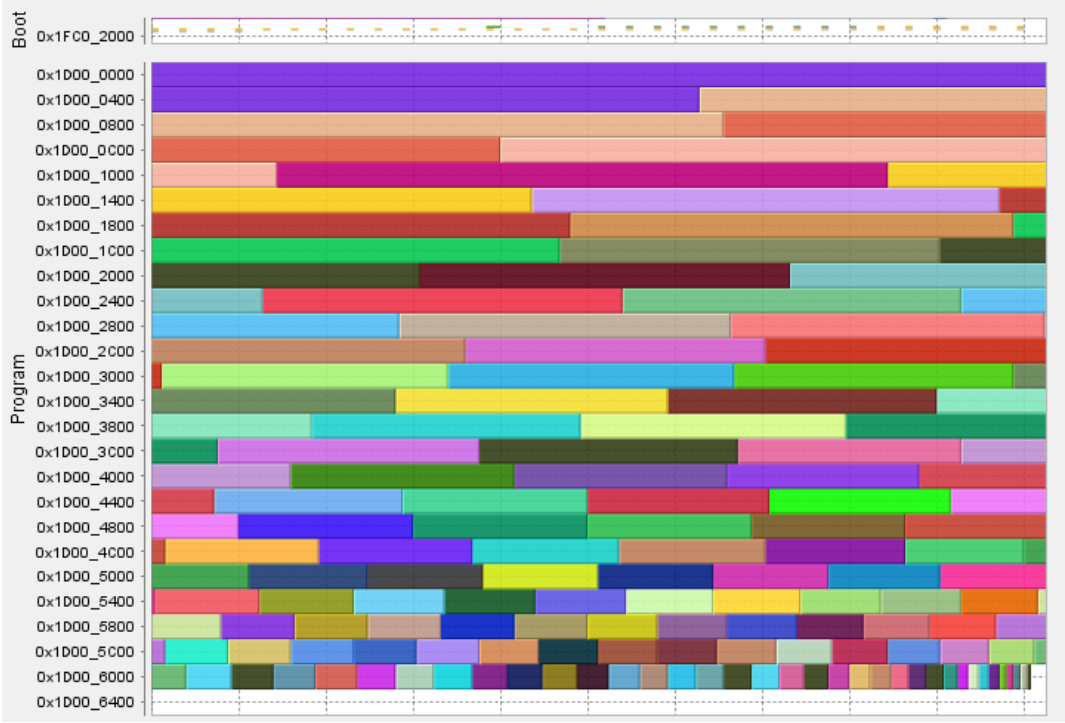
Optimizing the Bootloader

Provides information for optimizing the Bootloader.

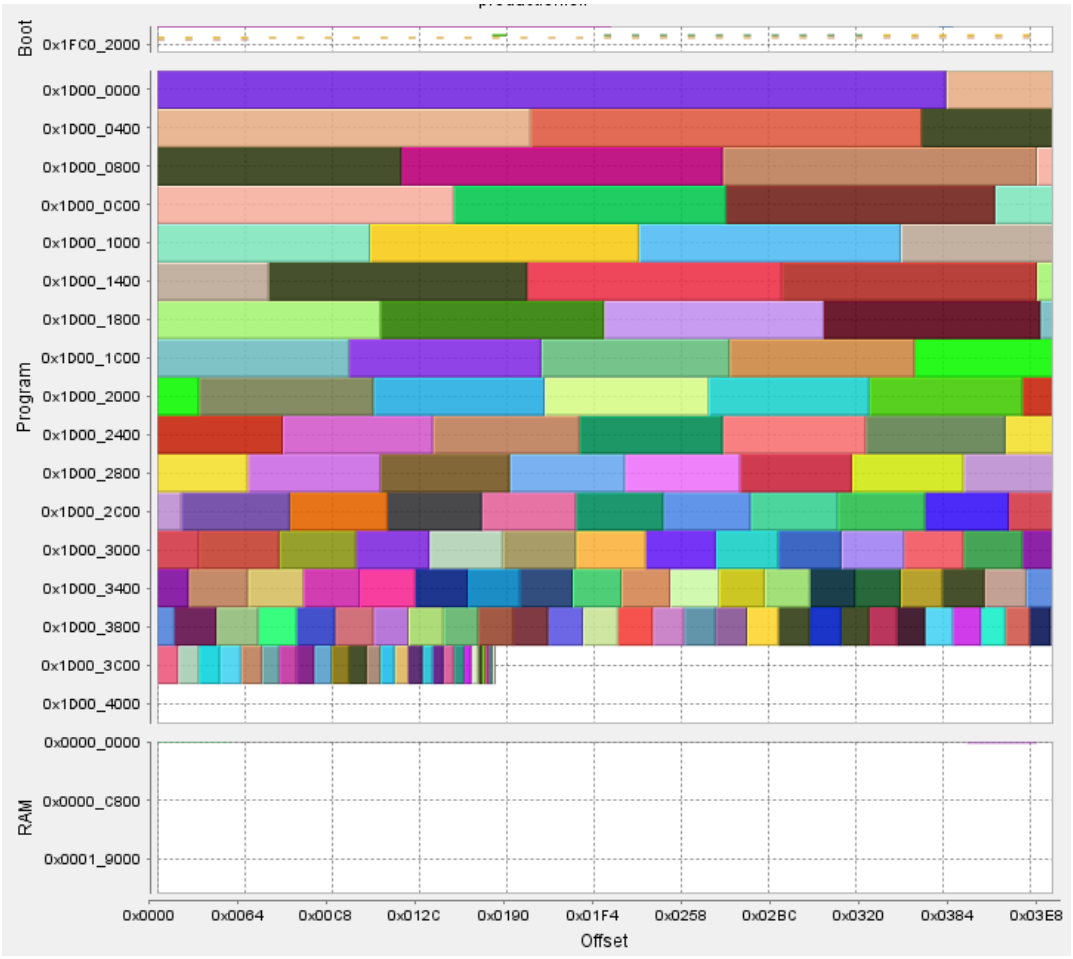
Description

By selecting a different optimization level, or by adding features to the Bootloader, the size of the Bootloader will change. These changes may require adjustments to the linker scripts of the Bootloader and the Application.

The following example will take a PIC32MX USB Device Bootloader, and set optimizations to save space. When the Bootloader is compiled with the default, `-O1`, optimization, it takes 24944 bytes of Flash. This can be seen in the following figure, which graphically shows the mapping of the Bootloader functions and Flash usage.



When the optimizations are set to `-Os`, and MIPS16 code is generated, the Bootloader code size shrinks to 15328 bytes, as shown in the following figure.



Because the Bootloader now ends at a lower address in Program Flash, the space can now be reclaimed for the application. Due to Flash page sizing, the application linker script must take the ending address of the Bootloader, and round it up to the start address of the next page. In this case, the new start address for the application can be 0x9D004000.

The MIPS32 core requires that Interrupt Vector Tables (IVTs) be placed on a 4 KB boundary, Therefore, the IVT for the Application can now be

placed at 0x9D005000, or any address above that in Program Flash that ends in 0x000.

The three lines to change in the application linker script are as follows, with the changes note in **bold** type:

Original Line of Code	Modified Line of Code
`_ebase_address = 0x9D008000;`	`_ebase_address = 0x9D00**5**000;`
`_RESET_ADDR = (0x9D007000);`	`_RESET_ADDR = (0x9D00**4**000);`
`kseg0_program_mem (rx) : ORIGIN = (0x9D000000 + 0x7000), LENGTH = 0x80000 - 0x7000`	`kseg0_program_mem (rx) : ORIGIN = (0x9D000000 + 0x**4**000), LENGTH = 0x80000 - 0x**4**000`

Library Interface

a) Functions

	Name	Description
	Bootloader_Initialize	Initializes the Bootloader Library.
	Bootloader_Tasks	Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

b) Data Types and Constants

	Name	Description
	MAJOR_VERSION	Bootloader Major Version Shown From a Read Version on PC
	MINOR_VERSION	Bootloader Minor Version Shown From a Read Version on PC
	BOOTLOADER_CLIENT_STATUS	Enumerated data type that identifies the Bootloader module client status.
	BOOTLOADER_INIT	A structure used to initialize the bootloader defining the different bootloader.
	BOOTLOADER_TYPE	A structure used to initialize the bootloader defining the different bootloader.
	BootInfo	This is variable BootInfo.
	BOOTLOADER_BUFFER	This is type BOOTLOADER_BUFFER.
	BOOTLOADER_CALLBACK	Pointer to a Bootloader callback function data type .
	BOOTLOADER_DATA_CALLBACK	Pointer to a Bootloader callback function data type .
	BOOTLOADER_BUFFER_SIZE	This is macro BOOTLOADER_BUFFER_SIZE.

Description

This section describes the Application Programming Interface (API) functions of the Bootloader Library.

Refer to each section for a detailed description.

a) Functions

Bootloader_Initialize Function

Initializes the Bootloader Library.

File

bootloader.h

C

```
void Bootloader_Initialize(const BOOTLOADER_INIT * drvBootloaderInit);
```

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function is used to initialize the Bootloader Library.

Remarks

This routine must be called before other Bootloader Library functions.

Preconditions

None.

Function

void Bootloader_Initialize (const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit);

Bootloader_Tasks Function

Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

File

bootloader.h

C

```
void Bootloader_Tasks();
```

Returns

None.

Description

This function maintains the Bootloader module state machine and manages the Bootloader Module object list items and responds to Bootloader Module events. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application.

Preconditions

None.

Example

```
while (true)
{
    Bootloader_Tasks ();

    // Do other tasks
}
```

Parameters

Parameters	Description
index	Object index for the specified module instance.

Function

void Bootloader_Tasks (SYS_MODULE_INDEX index);

b) Data Types and Constants

MAJOR_VERSION Macro

File

bootloader.h

C

```
#define MAJOR_VERSION 4    /* Bootloader Major Version Shown From a Read Version on PC */
```

Description

Bootloader Major Version Shown From a Read Version on PC

MINOR_VERSION Macro

File

bootloader.h

C

```
#define MINOR_VERSION 1    /* Bootloader Minor Version Shown From a Read Version on PC */
```

Description

Bootloader Minor Version Shown From a Read Version on PC

BOOTLOADER_CLIENT_STATUS Enumeration

Enumerated data type that identifies the Bootloader module client status.

File

bootloader.h

C

```
typedef enum {
  BOOTLOADER_CLIENT_STATUS_CLOSED,
  BOOTLOADER_CLIENT_STATUS_READY,
  BOOTLOADER_CLIENT_STATUS_WRITEFAILURE,
  BOOTLOADER_CLIENT_STATUS_WRITESUCCESS
} BOOTLOADER_CLIENT_STATUS;
```

Members

Members	Description
BOOTLOADER_CLIENT_STATUS_CLOSED	Client is closed or the specified handle is invalid
BOOTLOADER_CLIENT_STATUS_READY	Client is ready

Description

Bootloader Client Status

This enumeration defines the possible status of the Bootloader module client. It is returned by the () function.

Remarks

None.

BOOTLOADER_INIT Structure

A structure used to initialize the bootloader defining the different bootloader.

File

bootloader.h

C

```
typedef struct {
  BOOTLOADER_TYPE drvType;
  BOOTLOADER_STATES (* drvTrigger)(void);
} BOOTLOADER_INIT;
```

Description

Bootloader Initialization Type

This structure holds the bootloader types.

Remarks

None.

BOOTLOADER_TYPE Enumeration

A structure used to initialize the bootloader defining the different bootloader.

File

bootloader.h

C

```
typedef enum {
  TYPE_I2C,
  TYPE_USART,
  TYPE_USB_HOST,
  TYPE_USB_DEVICE,
  TYPE_ETHERNET_UDP_PULL,
  TYPE_SD_CARD
} BOOTLOADER_TYPE;
```

Description

Bootloader Type

This structure holds the bootloader types.

Remarks

None.

BootInfo Variable

File

bootloader.h

C

```
const uint8_t BootInfo[2] = { MINOR_VERSION, MAJOR_VERSION };
```

Description

This is variable BootInfo.

BOOTLOADER_BUFFER Union

File

bootloader.h

C

```
typedef union {
  uint8_t buffer[BOOTLOADER_BUFFER_SIZE + BOOTLOADER_BUFFER_SIZE];
  struct {
    uint8_t buff1[BOOTLOADER_BUFFER_SIZE];
    uint8_t buff2[BOOTLOADER_BUFFER_SIZE];
  } buffers;
} BOOTLOADER_BUFFER;
```

Description

This is type BOOTLOADER_BUFFER.

BOOTLOADER_CALLBACK Type

Pointer to a Bootloader callback function data type .

File

bootloader.h

C

```
typedef int (* BOOTLOADER_CALLBACK)(void);
```

Description

Bootloader General Callback Function Pointer

This data type defines a pointer to a Bootloader library callback function.

Remarks

This is used for callback on the following events:

- ERASE - Erase any additional areas (i.e. SQI, EBI, SPI memories)
- PROGRAM_COMPLETE - No more data coming
- START_APP - Anything that needs to be done prior to starting application
- BLANK_CHECK - Check external devices for erasure
- FORCE_BOOTLOAD - Any checks the bootloader wants to do before launching application

BOOTLOADER_DATA_CALLBACK Type

Pointer to a Bootloader callback function data type .

File

bootloader.h

C

```
typedef int (* BOOTLOADER_DATA_CALLBACK)(uint32_t address, uint32_t *data, uint32_t size);
```

Description

Bootloader Data to Program Callback Function Pointer

This data type defines a pointer to a Bootloader data callback function. This would be used to program data for an area not recognized by the bootloader as being in progam Flash (i.e. SPI Flash).

Remarks

This is used for callback on the following events:

- DATA_PROGRAM - Data to be programmed
- DATA_VERIFY - Compute the CRC for the given area

BOOTLOADER_BUFFER_SIZE Macro

File

bootloader.h

C

```
#define BOOTLOADER_BUFFER_SIZE 512
```

Description

This is macro BOOTLOADER_BUFFER_SIZE.

Files

Files

Name	Description
bootloader.h	The header file joins all header files used in the Bootloader Library and contains compile options and defaults.

Description

This section lists the source and header files used by the Bootloader Library.

bootloader.h

The header file joins all header files used in the Bootloader Library and contains compile options and defaults.

Enumerations

	Name	Description
	BOOTLOADER_CLIENT_STATUS	Enumerated data type that identifies the Bootloader module client status.
	BOOTLOADER_TYPE	A structure used to initialize the bootloader defining the different bootloader.

Functions

	Name	Description
	Bootloader_Initialize	Initializes the Bootloader Library.
	Bootloader_Tasks	Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

Macros

	Name	Description
	BOOTLOADER_BUFFER_SIZE	This is macro BOOTLOADER_BUFFER_SIZE.
	MAJOR_VERSION	Bootloader Major Version Shown From a Read Version on PC
	MINOR_VERSION	Bootloader Minor Version Shown From a Read Version on PC

Structures

	Name	Description
	BOOTLOADER_INIT	A structure used to initialize the bootloader defining the different bootloader.

Types

	Name	Description
	BOOTLOADER_CALLBACK	Pointer to a Bootloader callback function data type .
	BOOTLOADER_DATA_CALLBACK	Pointer to a Bootloader callback function data type .

Unions

	Name	Description
	BOOTLOADER_BUFFER	This is type BOOTLOADER_BUFFER.

Variables

	Name	Description
	BootInfo	This is variable BootInfo.

Description

Module for Microchip Bootloader Library

This header file includes all the header files required to use the Microchip Bootloader Library. Library features and options defined in the Bootloader Library configurations will be included in each build.

File Name

bootloader.h

Company

Microchip Technology Inc.

Class B Library Help

This section describes the Class B Library that is available in MPLAB Harmony.

Introduction

This library provides Class B Safety Software routines which can detect the occurrence of Faults in a single channel CPU.

Description

These routines have been developed in accordance with the IEC 60730 standard to support the Class B certification process. These routines can be directly integrated with the end user's application to test and verify the critical functionalities of a controller without affecting the end user's application. The Class B safety software routines can be called periodically at start-up or run time to test the following components:

- CPU Registers
- CPU Program Counter
- Invariable Memory
- Variable Memory
- Clock

NOTE: The term 'IEC 60730 standard' used in this document refers to the "IEC 60730-1 ed.5" Copyright © 2013 IEC, Geneva, Switzerland. www.iec.ch

Overview of the IEC 60730 Standard

Note: Microchip thanks the International Electrotechnical Commission (IEC) for permission to reproduce information from its International Standard IEC 60730-1ed.5 (2013). All such extracts are copyright of IEC, Geneva, Switzerland. All rights reserved. Further information on the IEC is available from www.iec.ch. IEC has no responsibility for the placement and con-text in which the extracts and contents are reproduced by the author, nor is IEC in any way responsible for the other content or accuracy therein.

The IEC 60730 standard defines the test and diagnostic methods that ensure the safe operation of the controlled equipment used in household appliances. Annex H of the IEC 60730 standard classifies the software into the following categories (see Appendix B: "IEC 60730-1 Table H.11.12.7"):

- Class A
- Class B
- Class C

The Class B Safety Software Library implements the important test and diagnostic methods that fall into the Class B category. These methods use various measures to detect and respond to the software-related Faults and errors.

According to the IEC 60730 standard, the controls with functions that fall into the Class B category should have one of the following structures:

- Single Channel with Functional Test - In this structure, the Functional test is executed prior to the application firmware execution
- Single Channel with Periodic Self-Test - In this structure, the Periodic tests are embedded within the firmware, and the self-test occurs periodically while the firmware is in Execution mode
- Dual Channel without Comparison - In this structure, two independent methods execute the specified operations

System Requirements

The following system requirements are recommended to run the Class B Safety Software Library:

- For the tests that require the independent time slot monitoring, the system hardware must be provided with at least two independent clock sources (e.g., crystal oscillator and line frequency).
- The user application determines whether the interrupts need to be enabled or disabled during the execution of the Class B Safety Software Library.

If an interrupt occurs during the execution of a Class B Safety Software Library routine, an unexpected change may occur in any of the registers. Therefore, when the Interrupt Service Routine (ISR) executes, the contents of the register will not match the expected content, and the ISR could return an incorrect result.

Using the Library

This topic describes the basic architecture of the Class B Library and provides information and examples on its use.

Description

Interface Header File: `classb.h`

The interface to the Class B Library is defined in the `classb.h` header file. Any C language source (`.c`) file that uses the Class B Library should include `classb.h`.

Library File:

The Class B Library archive (.a) file is installed with MPLAB Harmony.

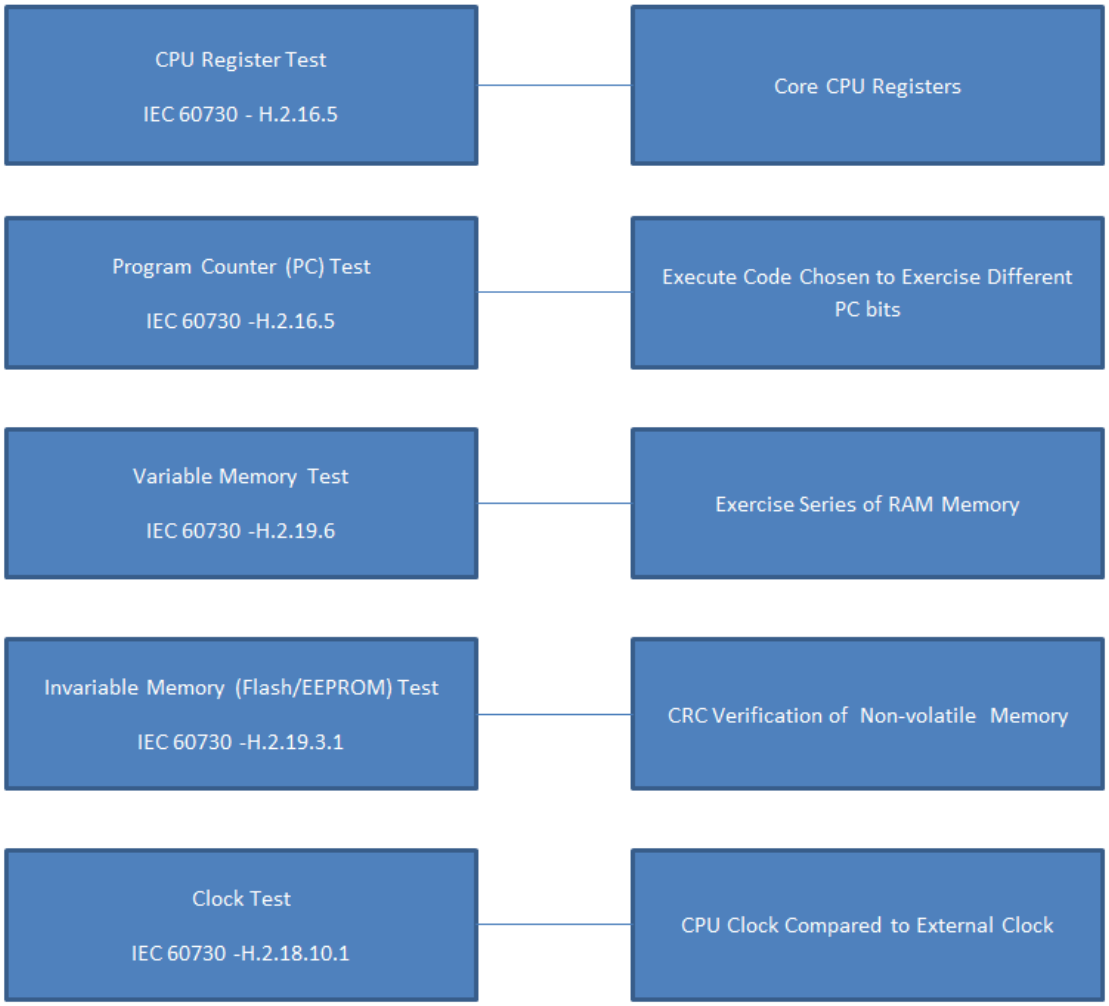
Please refer to the What is MPLAB Harmony? section for how the Class B Library interacts with the framework.

Abstraction Model

This library provides the low-level abstraction of the Class B Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Class B Software Abstraction Block Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Class B Library module. The Class B Safety Software Library includes several APIs, which are intended to maximize application reliability through Fault detection. These APIs help meet the IEC 60730 standard compliance.

Library Interface Section	Description
General Functions	Provides an error string function, which takes an error and converts it into human-readable text.

How the Library Works

This topic provides information on how the library works.

Description

CPU Register Test

The CPU Register test implements functional test H.2.16.5, as defined by the IEC 60730 standard. It detects stuck-at Faults in the CPU registers. This ensures that the bits in the registers are not stuck at a value of '0' or '1'.

The CPU Register test is a non-destructive test. It performs the following major tasks:

- The CPU registers and ghost registers are tested by first successively writing the binary sequences (length is dependent upon architecture), 010101... followed by 101010... into the registers, and then reading the values from these registers for verification.
- The test returns an error code if the returned values do not match.

Program Counter Test

The Program Counter (PC) test implements the functional test H.2.16.5 defined by the IEC 60730 standard. It detects stuck-at Faults in the PC. The PC holds the address of the next instruction to be executed. The test performs the following major tasks:

- Sets the error flag to the number of functions to be called.
- Calls the functions that are located in the Flash memory at different addresses chosen to alternate as many bits as possible in the PC.
- Each of these functions modifies the error flag.
- The error flag is tested after all functions are called.
- If the error flag is cleared, the PC has branched to all the correct locations.

Note: The size of the program memory varies by device. Thus the addresses used to test the PC will vary in order to test as many PC bits as possible. Refer to the specific device data sheet for more details.

Variable Memory Test

The Variable Memory test implements the Periodic Static Memory test H.2.19.6 defined by the IEC 60730 standard. It detects single bit Faults in variable memory. The variable memory contains data, which is intended to vary during program execution. The RAM Memory test is used to determine if any bit of the RAM memory is stuck at '1' or '0'. The March Memory test and Checkerboard test are some of the widely used static memory algorithms for checking the DC Faults. The following tests can be implemented using the Class B Safety Software Library:

- Checkerboard
- March C/C- Test
- March B Test
- March C Stack test

March Tests

A March test performs a finite set of operations on every memory cell in a memory array. Each operation performs the following tasks:

- Writes '0' to a memory cell (w0).
- Writes '1' to a memory cell (w1).
- Reads the expected value '0' from a memory cell (r0).
- Reads the expected value '1' from a memory cell (r1).

March Test Notations

> Arrange Addresses in ascending order

< Arrange Addresses in descending order

<> Arrange Addresses in either order

r0 Indicates a read operation (Read '0' from a memory cell)

r1 Indicates a read operation (Read '0' from a memory cell)

w0 Indicates a write operation (Write '0' to a memory cell)

w1 Indicates a write operation (Write '1' to a memory cell)

MARCH C/C- TEST

The March C/C- test is used to detect the following types of Fault in the variable memory:

• Stuck-at Fault

• Addressing Fault

• Transition Fault

• Coupling Fault

The complexity of the March C/C- test is 11n and 10n respectively, where n indicates the number of bits in the memory. This test can be run as either destructive or non-destructive. If run in non- destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory.

March C Algorithm

```
{
 <> (w0); > (r0, w1); > (r1, w0);
 <> (r0); < (r0, w1); < (r1, w0); > (r0)
```

```
}
```

Note1: is be removed for C-

MARCH B Test

The March B is a non -redundant test that can detect the following types of Fault:

- Stuck-at
- Linked Idempotent Coupling
- Inversion Coupling

This test is of complexity 17n, where n indicates the number of bits in the memory. This test can be run as either destructive or non-destructive. If run in non-destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory.

MARCH B ALGORITHM

```
{
<> (w0); > (r0, w1, r1, w0, r0, w1); > (r1, w0, w1);
 < (r1, w0, w1, w0); < (r0, w1, w0);
}
```

Checkerboard RAM Test

The Checkerboard RAM test writes the checkerboard patterns to a sequence of adjacent memory locations. This test is performed in units (memory chunks) of 8 bytes. This test can be run as either destructive or non-destructive. If run in non-destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory. This test performs the following major tasks:

- Saves the contents of the memory locations to be tested in the buffer defined by bufferAddress.
- Writes the binary value (length is dependent upon architecture) 101010... to the memory location, ‘N’, and the inverted binary value, 010101..., to the memory location, ‘N+1’, and so on, until the whole memory chunk is filled.
- Reads the contents of all the memory locations in the current chunk and verifies its contents. If the values match, the function continues; otherwise it stops and returns an error.
- Step 2 and 3 are repeated by writing the inverted pattern to the same locations.
- Once a memory chunk is completed the test of the next chunk is started until all of the requested memory area is tested.

Invariable Memory (Flash/EEPROM) Test

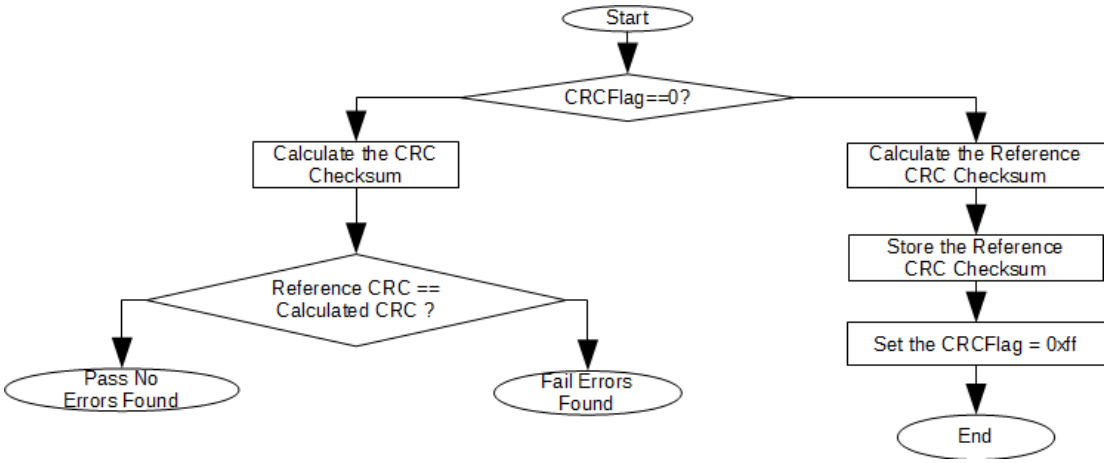
The Invariable Memory (Flash/EEPROM) test implements the periodic modified checksum H.2.19.3.1 defined by the IEC 60730 standard. It detects the single bit Faults in the invariable memory. The invariable memory in a system, such as Flash and EEPROM memory, contains data that is not intended to change during the program execution. The Flash/EEPROM Invariable Memory test computes the periodic check-sum using the Cyclic Redundancy Check (CRC-16). The CRC polynomial used to calculate the CRC-16 is shown below.

CRC-16 = 1 1000 0000 0000 0101 = 8005 (hex)

The CRC functions can be used to test the integrity of data stored in either Flash or EEPROM memory. This is done by calculating and returning the CRC value of the data stored in the location defined in the function call.

- If CRC_Flag is set to 0x00 at system start-up, the reference CRC checksum is computed.
- The reference checksum is stored in the Flash or EEPROM memory and the CRC flag is set to 0xFF.
- The CRC16 calculation function can be called periodically if the CRC flag is set to 0xFF.
- The checksum calculated from step 3 is compared with the reference checksum.
- If both values match, a status bit can be set by the user application to indicate that the invariable memory has passed the test and no errors were found.

Invariable Memory Test TEST Flowchart



Note: Other than the calls to CLASSB_CRCFlashTest and CLASSB_CRCEEPROMTest, all steps in this flowchart are to be executed by the application firmware.

Clock Test

According to the IEC 60730 standard, only harmonics and sub-harmonics of the clock need to be tested if using a quartz synchronized clock. The Clock test implements frequency monitoring H.2.18.10.1 as defined by the IEC 60730 standard. It verifies the reliability of the system clock (i.e., the system clock should be neither too fast nor too slow). The Clock Test function is used to verify the proper operation of the CPU clock. This test performs the following major tasks:

- The independent clock source (Sosc or T1OSC) is required for the test. The reference clock should be connected to a timer such as Timer1.
- During the test the number of CPU clock cycles each reference clock period is counted.
- If the number of clock cycles is outside a specified range, the function returns an error code.

Clock Test Using Line Frequency

The Clock Test function is used to verify the proper operation of the CPU clock. The 50 Hz/60 Hz line frequency is used as an independent clock source or a reference clock source. The input capture module is used for the period measurement. The 50 Hz/60 Hz line frequency is fed to the Input Capture pin (IC1) of the respective device. This test performs the following major tasks:

- The IC1 input is used as the independent clock source/reference clock source to capture the hardware Timer2. An external reference frequency, usually the line frequency, has to be applied to the IC1 input pin.
- The Input Capture 1 is configured as follows:
- Timer2 is selected as IC1 time base
- Capture is performed on every rising edge
- Capture done event is generated on every second capture.
- The hardware Timer2 prescaler is calculated (based on the input reference frequency and the current PB frequency) as being the smallest divider possible such that the 16 bit Timer2 does not overflow within a period time of the input reference signal. This way, the best resolution is achieved for the given conditions. If no valid pre-scaler value can be obtained an error value is returned.
- The IC1 performs the capture on every rising edge of the input reference frequency. For period measurement, the capture done event is generated after the IC1 module takes two time stamps i.e. after every period of the input reference (20 ms if reference frequency is 50 Hz, 16.66ms if the reference frequency is 60 Hz).
- Once the capture done event is signaled, the two Timer2 captured readings are extracted and the number of elapsed PB clocks is calculated as being the difference between the two readings. If this value crosses the defined boundary limits the function returns an appropriate error value

Calculation Example 1:

System Clock = 80 MHz

PB Clock = 80 MHz (PB divider =1:1)

Input Reference = 50 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 25

Actual T2 Divider = 32.

The number of cycles counted in the Reference clock period is = (80,000,000/32)/50 = 50,000.

Calculation Example 2:

System Clock = 80 MHz

PB Clock = 10 MHz (PB divider =1:8)

Input Reference = 60 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 3

Actual T2 Divider = 4.

The number of cycles counted in the Reference clock period is = (10,000,000/4)/60 = 41,666.

Configuring the Library

The configuration of the Class B Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Class B Library. Based on the selections made, the Class B Library may support the selected features. These configuration settings will apply to all instances of the Class B Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Class B Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/classb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
classb.h	Includes all MPLAB Harmony-compatible function calls for the Class B Library.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
<file name>	<add description>

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
<file name>	<add description>

Module Dependencies

The Class B Library does not depend on any other modules.

Library Interface

a) Functions

	Name	Description
	CLASSB_ClockLineFreqTest	The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.
	CLASSB_ClockTest	The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.
	CLASSB_CPUPCTest	The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CPURegistersTest	The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CRCFlashTest	The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.
	CLASSB_RAMCheckerBoardTest	The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchBTest	The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCStackTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

	Flash_CRC16	• Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize) * • PreCondition: pBuff valid pointer * • Input: - pBuff: buffer to calculate CRC over • - crcPoly: the generator polynomial to be used • - crcReg: initial value of the CRC LFSR (seed) • - bSize: buffer size, bytes * • Output: value of the CRC * • Side Effects: None * • Overview: Shifts bytes through the CRC shift register. * • Note: Simple (and slow) CRC16 calculation directly based • on the hardware LFSR implementation.

b) Data Types and Constants

	Name	Description
	CLASSBRESULT	This is type CLASSBRESULT.
	CLASSB_RAM_TEST_CYCLE_SIZE	This is macro CLASSB_RAM_TEST_CYCLE_SIZE.
	CLOCK_TEST_INTERRUPT_SOURCE	This is macro CLOCK_TEST_INTERRUPT_SOURCE.
	CLOCK_TEST_REFERENCE_FREQ	Defines the reference frequency that is applied to the SOSC input.
	CLOCK_TEST_TIMER	This is macro CLOCK_TEST_TIMER.
	CRC_08_GEN_POLY	Flash CRC16 Test Generator Polynomials The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation. Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.
	CRC_16_GEN_POLY	This is macro CRC_16_GEN_POLY.
	CRC_32_GEN_POLY	This is macro CRC_32_GEN_POLY.
	CRC_CCITT_GEN_POLY	This is macro CRC_CCITT_GEN_POLY.
	FLASH_CRC16_MASK	mask to retain the useful CRC result for 16 bit polynomial
	FLASH_CRC16_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC32_MASK	mask to retain the useful CRC result for 32 bit polynomial
	FLASH_CRC32_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC8_MASK	mask to retain the useful CRC result for 8 bit polynomial
	FLASH_CRC8_MSB	mask to obtain the MSb-1, transport to the MSb.
	USE_MARCHC_MINUS	This is macro USE_MARCHC_MINUS.

Description

This section describes the Application Programming Interface (API) functions of the Class B Library.

Refer to each section for a detailed description.

a) Functions

CLASSB_ClockLineFreqTest Function

The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_ClockLineFreqTest(unsigned int clockFrequency, unsigned int lineFrequency, unsigned int tolerance);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The monitored CPU clock is within the requested limits.

CLASSB_TEST_FAIL - The test failed. The monitored CPU clock is greater than the specified upper limit; The monitored CPU clock is less than the specified lower limit; The frequency of the provided reference was too low and could not be used.

Description

The CPU Clock Line Test verifies that the system clock is within specified limits. An external frequency applied on Input Capture 1 pin (IC1) is used as the reference clock. The hardware Timer2 that runs on the system Peripheral Bus (PB) clock is used to monitor the CPU clock and Peripheral Bus divider.

The test performs the following major steps:

1. The IC1 input is used as the independent clock source/reference clock source to capture the hardware Timer2. An external reference frequency, usually the line frequency, has to be applied to the IC1 input pin.
2. The Input Capture 1 is configured as follows:

- Timer2 is selected as IC1 time base
- Capture is performed on every rising edge
- Capture done event is generated on every second capture.

3. The hardware Timer2 pre-scaler is calculated (based on the input reference frequency and the current PB frequency) as being the smallest divider possible such that the 16 bit Timer2 does not overflow within a period time of the input reference signal. This way, the best resolution is achieved for the given conditions. If no valid pre-scaler value can be obtained an error value is returned.
4. The IC1 performs the capture on every rising edge of the input reference frequency. For period measurement, the capture done event is generated after the IC1 module takes two time stamps i.e. after every period of the input reference (20 ms if reference frequency is 50 Hz, 16.66ms if the reference frequency is 60 Hz).
5. Once the capture done event is signalled, the 2 Timer2 captured readings are extracted and the number of elapsed PB clocks is calculated as being the difference between the two readings. If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

Calculation example 1: System Clock = 80 MHz PB Clock = 80 MHz (PB divider =1:1) Input Reference = 50 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 25 Actual T2 Divider = 32. The number of cycles counted in the Reference clock period is = (80,000,000/32)/50 = 50,000.

Calculation example 2: System Clock = 80 MHz PB Clock = 10 MHz (PB divider =1:8) Input Reference = 60 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 3 Actual T2 Divider = 4. The number of cycles counted in the Reference clock period is = (10,000,000/4)/60 = 41,666.

Remarks

The test uses the hardware Input Capture 1 module. It initializes the module as needed and, after the test is done, it shuts it off. The previous state of IC1 is not preserved/restored.

The test uses the hardware Timer2. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The value of the PB frequency which is used as input by the Timer2 is derived from the system CPU clock by dividing it with the PB divider. The test does not change the value of the PB divider, it uses the current value.

The interrupts should be disabled when executing this test as the time spent in ISR's affects the accurate timing of this routine.

The frequency of signal used as a reference on IC1 input should normally be the line frequency. However, any frequency can be used as long as a a valid Timer2 divider can be obtained (see the example calculation below for the 16 bit Timer2). If the reference frequency is too low, a valid divider for the Timer 2 won't be possible (the maximum divider for Timer2 is 256). You can go to a greater PB divider in this case. If the selected reference frequency is too high the number of Timer2 captured counts will be too small and the measurement won't have enough resolution.

Preconditions

None.

Example

```
int testRes=CLASSB_ClockLineFreqTest(80000000, 60, 80000, 100000);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process CPU clock high failure
}
```

Parameters

Parameters	Description
clockFrequency	the current system running frequency, Hz
lineFrequency	the frequency of the reference applied to the IC1 input pin, Hz. Usual values are 50/60 Hz
tolerance	the tolerance level of the system oscillator as a percentage (0 - 100).

Function

CLASSBRESULT CLASSB_ClockLineFreqTest(unsigned int clockFrequency, unsigned int lineFrequency, uint8_t tolerance)

CLASSB_ClockTest Function

The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_ClockTest(unsigned int clockFrequency, unsigned int referenceFrequency, unsigned int testLengthMsec, unsigned int tolerance);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The monitored CPU clock is within the requested limits.

CLASSB_TEST_FAIL - The test failed. The monitored CPU clock is greater than the specified upper limit; The monitored CPU clock is less than the specified lower limit.

Description

The CPU Clock test verifies that the system clock is within specified limits. The secondary oscillator (SOSC) is used as the reference clock. The CPU Core Timer that runs on the CPU system clock is monitored.

The test performs the following major steps:

1. The LP SOSC is used as the independent clock source/reference clock source connected to hardware Timer1.
2. The CPU Core Timer monitored in this measurement is incremented every other CPU system clock. Usually the system runs on the Primary oscillator with PLL as the clock source to the CPU. However, any clock source except the SOSC itself which is used as a reference is valid for this test.
3. Timer1 is configured to time out after the specified interval of time elapsed (e.g. 10 ms).
4. The content of the Core Timer is saved at the beginning of the measurement, once the Timer 1 is started.
5. When the hardware Timer1 times out another reading of the Core Timer is taken and the difference is made with the start value. This difference value represents the number of CPU clock cycles counted by the Core Timer during the SOSC period of time.
6. If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

Remarks

The test uses the hardware Timer1. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The test assumes that the Core Timer is enabled and linearly counting up as it should do during normal system operation. If your code specifically disables the Core Timer, it should enable it before this test is called. If the value in the Core Timer is updated/changed as part of an ISR, this ISR should be disabled.

The interrupts should be disabled when executing this test as the time spent in ISR's affects the accurate timing of this routine.

The SOSC is used as a reference. Use the CLOCK_TEST_SOSC_FREQ for adjustments to this value, if needed.

The value of the CPU clock monitoring time, nMs, is limited because of the use of the hardware Timer1 which is a 16 bit timer. Therefore, the value loaded into this Timer1 register should not exceed 2^16-1.

Preconditions

None.

Example

```
int testRes=CLASSB_ClockTest(80000000, 100, 80000, 100000);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
```

```
else
{
    // process CPU clock high failure
}
```

Parameters

Parameters	Description
clockFrequency	the current system running frequency, Hz
referenceFrequency	the frequency of the reference applied to the SOSC pin, Hz.
testLengthMsec	number of milliseconds to be used for the CPU Clock monitoring. 1 <= testLengthMsec <= 1000
tolerance	the tolerance level of the system oscillator as a percentage (0 - 100).

Function

CLASSBRESULT CLASSB_ClockTest(unsigned int sysClk, int nMs, unsigned int hiClkErr, unsigned int loClkErr)

CLASSB_CPUPCTest Function

The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_CPUPCTest();
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The PC register holds the correct address.

CLASSB_TEST_FAIL - The test failed. The PC register has been detected to hold an incorrect address.

Description

The Program Counter test is a functional test of the PC. It checks that the PC register is not stuck and it properly holds the address of the next instruction to be executed.

The tests performs the following major tasks:

1. The Program Counter test invokes functions that are located in memory at different addresses.
2. That all of the functions is executed is verified.

Remarks

The test uses 3 different functions: CLASSB_CPUPCTestFunction1() CLASSB_CPUPCTestFunction2() CLASSB_CPUPCTestFunction3()

Preconditions

None.

Example

```
int testRes=CLASSB_CPUPCTest();
if(testRes==PC_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Function

int CLASSB_CPUPCTest (void)

CLASSB_CPURegistersTest Function

The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_CPURegistersTest();
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. CPU registers have not been detected to have stuck bits.

CLASSB_TEST_FAIL - The test failed. Some CPU register(s) has been detected to have stuck bits.

Description

This routine detects stuck-at Faults in the CPU registers. This ensures that the bits in the registers are not stuck at a value ?0? or ?1?.

Remarks

This is a non-destructive test.

Interrupts should be disabled when calling this test function.

Preconditions

None.

Example

```
int testRes=CLASSB_CPURegistersTest();
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Function

int CLASSB_CPURegistersTest (void)

CLASSB_CRCFlashTest Function

The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
unsigned int CLASSB_CRCFlashTest(char* startAddress, char* endAddress, unsigned int crcPoly, unsigned int crcSeed);
```

Returns

The value of the calculated CRC over the specified memory area.

Description

This routine detects the single bit Faults in the invariable memory. The invariable memory in a system, such as Flash and EEPROM memory, contains data that is not intended to vary during the program execution.

The test calculates the 16 bit CRC of the supplied memory area using the standard LFSR (Linear Feedback Shift Register) implementation. It calculates over the memory area between the startAddress and endAddress and returns the CRC Value. The 16 bit CRC is calculated using the supplied generator polynomial and initial seed. Different generator polynomials can be used as indicated above.

Remarks

This is a non-destructive memory test.

The startAddress and endAdress over which the CRC value is calculated are PIC32 variant and application dependent. They are run-time parameters.

Preconditions

None.

Example

```
unsigned int crcRes=CLASSB_CRCFlashTest(startAddress, endAddress, CRC_16_GEN_POLY, 0xffff);
if(crcRes==prevCalculatedCrc)
{
    // process test success
}
else
{
    // process tests failure: the CRC of the memory area has changed.
}
```

Parameters

Parameters	Description
startAddress	start Address of the memory area to start CRC calculation from
endAddress	final address for which the CRC is calculated
crcPoly	the generator polynomial to be used. One of the standard supplied polynomials can be used as well as other user defined ones.
crcSeed	the initial value in the CRC LFSR. The usual recommended value is 0xffff.

Function

unsigned int CLASSB_CRCFlashTest(char* startAddress, char* endAddress, unsigned int crcPoly, unsigned int crcSeed)

CLASSB_RAMCheckerBoardTest Function

The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMCheckerBoardTest(unsigned int * startAddress, unsigned int length);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have stuck bits.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have stuck bits.

Description

This routine detects single bit Faults in the variable memory. This ensures that the bits in the tested RAM are not stuck at a value ?0? or ?1?.

The test writes the checkerboard pattern (0x55555555 followed by 0xaaaaaaaa) to adjacent memory locations starting at ramStartAddress. It performs the following steps:

1. A temporary memory buffer (stack) is tested for reliability.
1. The content of a 64 bytes memory chunk to be tested is saved in temporary memory buffer.
2. Writes the pattern 0x55555555 followed by 0xaaaaaaaa to adjacent memory locations filling up the 64 bytes memory chunk.
3. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result and go to step 4. Else set the error result and go to step 6.
4. Writes the inverted pattern 0xaaaaaaaa followed by 0x55555555 to adjacent memory locations filling up the 64 bytes memory chunk.
5. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result. Else set the error result.
6. The content of the tested 64 bytes memory chunk is restored from the temporary memory buffer.
7. If the result shows error the test is done and returns.
8. The address pointer is incremented to point to the next sequential 64 bytes memory chunk and the test is repeated from step 1 until all the number of requested memory locations is tested.

Remarks

This is a non-destructive memory test. The content of the tested memory area is saved and restored. The test operates in 64 bytes long memory chunks at a time.

At least 32 bytes should be available for stack for executing the RAM Checker Board test. The tested RAM area must not overlap the stack.

The Start Address from which the Checker Board test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMCheckerBoardTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the checker Board test is to be performed Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 64.

Function

int CLASSB_RAMCheckerBoardTest (int* ramStartAddress, int ramSize)

CLASSB_RAMMarchBTest Function

The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchBTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have faults.

Description

This test is a complete and non redundant test capable of detecting stuck-at, linked idempotent coupling or Inversion Coupling faults. This test is of complexity 17n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The address must be properly word aligned and the length of the area to be tested must be an integral multiple of the data width access.

Remarks

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 100 bytes should be available for stack for executing the March B test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March B test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Refer to the AN1229 for details regarding the CLASSB_RAMMarchBTest() and the Class B Software Library.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchBTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
```

```
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the March B test is to be performed Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 4.

Function

int CLASSB_RAMMarchBTest (int* ramStartAddress, int ramSize)

CLASSB_RAMMarchCStackTest Function

The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchCStackTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM and Stack area tested have not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Either some RAM or Stack area location has been detected to have faults,

Description

This test is a complete and non redundant test capable of detecting stuck-at, addressing, transition and coupling faults. This test is of complexity 11n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The addresses must be properly word aligned and the lengths of the areas to be tested must be an integral multiple of the data width access.

Remarks

This function is just a helper to March C test both a regular RAM area and a Stack area. First the RAM area is tested using the standard March C test. If the test succeeded, the requested Stack area is saved/copied in the RAM area that has just been tested and then the March C test is run over the Stack area as if it were a regular RAM area. The saved Stack area is restored and the result of the test is returned to the user.

The RAM and Stack areas have to NOT overlap! The Stack grows downwards so the tested area is: [stackTopAddress-stackSize, stackTopAddress] Also the size of the Stack area to be tested has to be less than the size of the RAM area.

The processor SP register is changed to the point to the RAM area while the Stack area is tested. Since running the MARC C test requires at least 128 bytes of stack, the RAM area size should be at least 128 bytes long. Once the Stack area is tested, the SP register is restored.

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 128 bytes should be available for stack for executing the March C test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March C test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchCStackTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
```

```
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address of RAM area to be used as temporary stack. The stack will be copied into this memory, and the stack pointer redirected here. Has to NOT overlap the Stack area! Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The portion of the stack to be tested will be from the Stack Start (Highest address) to this address minus ramSize. This value also defines the amount of memory available as a temporary stack. The size must be a number multiple of 4. The size of the RAM area tested has to be >= 128 bytes.

Function

CLASSBRESULT CLASSB_RAMMarchCStackTest (unsigned int* ramStartAddress,

unsigned int ramSize)

CLASSB_RAMMarchCTest Function

The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchCTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have faults.

Description

This test is a complete and non redundant test capable of detecting stuck-at, addressing, transition and coupling faults. This test is of complexity 11n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The address must be properly word aligned and the length of the area to be tested must be an integral multiple of the data width access.

Remarks

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 100 bytes should be available for stack for executing the March C test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March C test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchCTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the March C test is to be performed Must be properly 32 bit aligned.

ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 4.

Function

CLASSBRESULT CLASSB_RAMMarchCTest(unsigned int* ramStartAddress,
unsigned int ramSize)

Flash_CRC16 Function

File

classb_config_template.h

C

```
unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int
bSize);
```

Description

- Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize)
*
- PreCondition: pBuff valid pointer
*
- Input: - pBuff: buffer to calculate CRC over
- - crcPoly: the generator polynomial to be used
- - crcReg: initial value of the CRC LFSR (seed)
- - bSize: buffer size, bytes
*
- Output: value of the CRC
*
- Side Effects: None
*
- Overview: Shifts bytes through the CRC shift register.
*
- Note: Simple (and slow) CRC16 calculation directly based
- on the hardware LFSR implementation.

b) Data Types and Constants

CLASSBRESULT Enumeration

File

classb_config_template.h

C

```
typedef enum {
  CLASSB_TEST_PASS = 0,
  CLASSB_TEST_FAIL,
  CLASSB_TEST_TIMEOUT,
  CLASSB_TEST_INPROGRESS
} CLASSBRESULT;
```

Description

This is type CLASSBRESULT.

CLASSB_RAM_TEST_CYCLE_SIZE Macro

File

classb_config_template.h

C

```
#define CLASSB_RAM_TEST_CYCLE_SIZE 64
```

Description

This is macro CLASSB_RAM_TEST_CYCLE_SIZE.

CLOCK_TEST_INTERRUPT_SOURCE Macro

File

classb_config_template.h

C

```
#define CLOCK_TEST_INTERRUPT_SOURCE INT_SOURCE_TIMER_1
```

Description

This is macro CLOCK_TEST_INTERRUPT_SOURCE.

CLOCK_TEST_REFERENCE_FREQ Macro

Defines the reference frequency that is applied to the SOSC input.

File

classb_config_template.h

C

```
#define CLOCK_TEST_REFERENCE_FREQ 32768
```

Description

CPU Clock Test Secondary Oscillator (SOSC) reference frequency

This definition is used for internal calculation in the CLASSB_ClockTest() function. Normally this input is driven by a 32.768 KHz xtal. You can adjust this value if the need occurs (temperature drifts, etc).

CLOCK_TEST_TIMER Macro

File

classb_config_template.h

C

```
#define CLOCK_TEST_TIMER TMR_ID_1
```

Description

This is macro CLOCK_TEST_TIMER.

CRC_08_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_08_GEN_POLY 0x07     /* x^8 + x^2 + x + 1*/
```

Description

Flash CRC16 Test Generator Polynomials

The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation.

Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.

CRC_16_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_16_GEN_POLY 0x8005   /* x^16 + x^15 + x^2 + 1*/
```

Description

This is macro CRC_16_GEN_POLY.

CRC_32_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_32_GEN_POLY 0x04C11DB7  /* x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 +
x^5 + x^4 + x^2 + x + 1*/
```

Description

This is macro CRC_32_GEN_POLY.

CRC_CCITT_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_CCITT_GEN_POLY 0x1021   /* x^16 + x^12 + x^5 + 1*/
```

Description

This is macro CRC_CCITT_GEN_POLY.

FLASH_CRC16_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC16_MASK ((1L<<16)-1)            // mask to retain the useful CRC result for 16 bit
polynomial
```

Description

mask to retain the useful CRC result for 16 bit polynomial

FLASH_CRC16_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC16_MSB (1L<<(16-1))            // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

FLASH_CRC32_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC32_MASK ((1L<<32)-1)            // mask to retain the useful CRC result for 32 bit
polynomial
```

Description

mask to retain the useful CRC result for 32 bit polynomial

FLASH_CRC32_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC32_MSB (1L<<(32-1))            // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

FLASH_CRC8_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC8_MASK ((1L<<8)-1)              // mask to retain the useful CRC result for 8 bit
polynomial
```

Description

mask to retain the useful CRC result for 8 bit polynomial

FLASH_CRC8_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC8_MSB (1L<<(8-1))              // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

USE_MARCHC_MINUS Macro

File

classb_config_template.h

C

```
#define USE_MARCHC_MINUS
```

Description

This is macro USE_MARCHC_MINUS.

Files

Files

Name	Description
classb_config_template.h	Class B interface header for definitions common to the Class B Library.

Description

This section lists the source and header files used by the Class B Library.

classb_config_template.h

Class B interface header for definitions common to the Class B Library.

Enumerations

	Name	Description
	CLASSBRESULT	This is type CLASSBRESULT.

Functions

	Name	Description
	CLASSB_ClockLineFreqTest	The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.
	CLASSB_ClockTest	The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.
	CLASSB_CPUPCTest	The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CPURegistersTest	The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CRCFlashTest	The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.
	CLASSB_RAMCheckerBoardTest	The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchBTest	The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCStackTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	Flash_CRC16	• Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize) * • PreCondition: pBuff valid pointer * • Input: - pBuff: buffer to calculate CRC over • - crcPoly: the generator polynomial to be used • - crcReg: initial value of the CRC LFSR (seed) • - bSize: buffer size, bytes * • Output: value of the CRC * • Side Effects: None * • Overview: Shifts bytes through the CRC shift register. * • Note: Simple (and slow) CRC16 calculation directly based • on the hardware LFSR implementation.

Macros

	Name	Description
	CLASSB_RAM_TEST_CYCLE_SIZE	This is macro CLASSB_RAM_TEST_CYCLE_SIZE.
	CLOCK_TEST_INTERRUPT_SOURCE	This is macro CLOCK_TEST_INTERRUPT_SOURCE.
	CLOCK_TEST_REFERENCE_FREQ	Defines the reference frequency that is applied to the SOSC input.
	CLOCK_TEST_TIMER	This is macro CLOCK_TEST_TIMER.
	CRC_08_GEN_POLY	Flash CRC16 Test Generator Polynomials The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation. Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.
	CRC_16_GEN_POLY	This is macro CRC_16_GEN_POLY.
	CRC_32_GEN_POLY	This is macro CRC_32_GEN_POLY.
	CRC_CCITT_GEN_POLY	This is macro CRC_CCITT_GEN_POLY.
	FLASH_CRC16_MASK	mask to retain the useful CRC result for 16 bit polynomial
	FLASH_CRC16_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC32_MASK	mask to retain the useful CRC result for 32 bit polynomial
	FLASH_CRC32_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC8_MASK	mask to retain the useful CRC result for 8 bit polynomial
	FLASH_CRC8_MSB	mask to obtain the MSb-1, transport to the MSb.
	USE_MARCHC_MINUS	This is macro USE_MARCHC_MINUS.

Description

Class B Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Class B Library for all families of Microchip microcontrollers. The definitions in this file are common to the Class B Library.

File Name

classb.h

Company

Microchip Technology Inc.

Crypto Library Help

This section describes the Cryptographic (Crypto) Library that is available in MPLAB Harmony.

Introduction

This library provides a software Cryptographic Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Cryptographic Library includes functions to perform encryption, decryption, hashing, authentication, and compression within the embedded application. Random number generation (RNG) functions are also provided.

Block Ciphers

The library provides DES, 3DES, and AES for block cipher needs. Depending on the algorithm in use, CBC and CTR modes are supported.

Public Key Cryptography

The library provides RSA and Elliptic Curve Cryptography (ECC) for Public Key Cryptography, and Diffie-Hellman (DH) for key agreement arrangements.

Hash Functions

The library provides MD5, SHA, SHA-256, SHA-384, and SHA-512 for hashing. These functions do not require keys or initialization vectors (IV).

Random Number Generation Functions

The library provides functions to generate either a single pseudo-random number, or a block of such numbers.

Using the Library

This topic describes the basic architecture of the Cryptographic (Crypto) Library and provides information and examples on its use.

Description

Interface Header File: `crypto.h`

The interface to the Crypto Library is defined in the `crypto.h` header file. Any C language source (`.c`) file that uses the Crypto Library should include `crypto.h`.

Library File:

The Crypto Library archive (`.a`) file is installed with MPLAB Harmony.

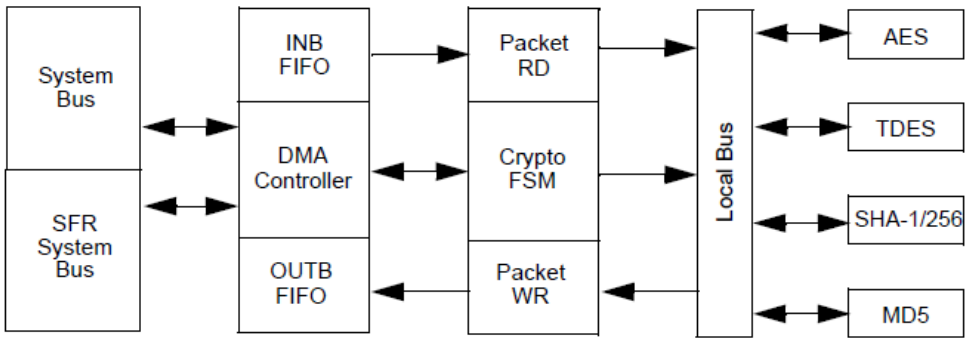
Please refer to the What is MPLAB Harmony? section for how the Crypto Library interacts with the framework.

Abstraction Model

This library provides the low-level abstraction of the Cryptographic Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Cryptographic (Crypto) Software Abstraction Block Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Cryptographic (Crypto) Library module.

Library Interface Section	Description
General Functions	Provides an error string function, which takes an error and converts it into human-readable text.
Compression Functions	Provides Huffman compression and decompression functions.
MD5 Functions	Provides data add, finalize, and initialize MD5 functions.
Random Number Generator Functions	Provides get, initialize, and block generate RNG functions.
AES Encryption/Decryption Functions	Provides AES encryption and decryption functions.
ECC Encryption/Decryption Functions	Provides ECC encryption and decryption functions.
RSA Encryption/Decryption Functions	Provides RSA encryption and decryption functions.
Triple DES Encryption/Decryption Functions	Provides 3DES encryption and decryption functions.
HMAC Hash Functions	Provides HMAC data add, finalize, and set key Hash functions.
SHA Hash Functions	Provides SHA data add, finalize, and initialize Hash functions.
SHA256 Hash Functions	Provides SHA256 data add, finalize, and initialize Hash functions.
SHA384 Hash Functions	Provides SHA384 data add, finalize, and initialize Hash functions.
SHA512 Hash Functions	Provides SHA512 data add, finalize, and initialize Hash functions.

Configuring the Library

The configuration of the Cryptographic Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Cryptographic Library. Based on the selections made, the Cryptographic Library may support the selected features. These configuration settings will apply to all instances of the Cryptographic Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Crypto Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/crypto`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`crypto.h`	Includes all MPLAB Harmony-compatible function calls for the Crypto Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
<install-dir>/build/framework/crypto/crypto.X	Include this project to bring in the library and all associated functions. There are two build configuration options available for this library: • SW – Software-only configuration for all Microchip PIC32 microcontrollers. • HW – Hardware acceleration configuration for PIC32 devices that have a hardware encryption module For both configurations, adjust the properties if 16-bit code is desired, and set the appropriate version of microcontroller.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
<install-dir>/build/framework/crypto/zlib.X	This project provides the code for Huffman compression in the Crypto Library. If the application does not use Huffman compression, this library may be excluded. There are two build configuration options available for this library: • PIC32MX – for PIC32MX devices • PIC32MZ – for PIC32MZ devices For both configurations, adjust the properties if 16-bit code is desired.

Module Dependencies

The Crypto Library does not depend on any other modules.

Library Interface

a) General Functions

	Name	Description
	CRYPT_RNG_Initialize	Initializes the random number generator.
	CRYPT_ERROR_StringGet	Reports the nature of an error.

b) Compression Functions

	Name	Description
	CRYPT_HUFFMAN_Compress	Compresses a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.

c) MD5 Functions

	Name	Description
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
	CRYPT_MD5_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

d) Random Number Generator Functions

	Name	Description
	CRYPT_RNG_BlockGenerate	Createa several random numbers.
	CRYPT_RNG_Get	Gets one random number.

e) AES Encryption/Decryption Functions

	Name	Description
	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.

	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.

f) ECC Encryption/Decryption Functions

	Name	Description
	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
	CRYPT_ECC_DSA_HashSign	Signs a message digest.
	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
	CRYPT_ECC_KeySizeGet	Returns the key size in octets.
	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.

g) RSA Encryption/Decryption Functions

	Name	Description
	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
	CRYPT_RSA_Free	Releases the memory used for the key and clean up the context.
	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.
	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.

h) Triple DES (3DES) Encryption/Decryption Functions

	Name	Description
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

i) HMAC Hash Functions

	Name	Description
	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.

j) SHA Hash functions

	Name	Description
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

k) SHA256 Hash Functions

	Name	Description
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

l) SHA384 Hash Functions

	Name	Description
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.

m) SHA512 Hash Functions

	Name	Description
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.

n) Data Types and Constants

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES
	MC_CRYPTO_API_H	Defines Microchip CRYPTO API layer

Description

This section describes the Application Programming Interface (API) functions of the Cryptographic Library.

Refer to each section for a detailed description.

a) General Functions

CRYPT_RNG_Initialize Function

Initializes the random number generator.

File

crypto.h

C

```
int CRYPT_RNG_Initialize(CRYPT_RNG_CTX*);
```

Returns

- negative - An error occurred setting up the random number generator.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context that stores information relative to random number generation.

Preconditions

None.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
```

```
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to random number generator context.

Function

int CRYPT_RNG_Initialize(CRYPT_RNG_CTX* rng)

CRYPT_ERROR_StringGet Function

Reports the nature of an error.

File

crypto.h

C

```
int CRYPT_ERROR_StringGet(int, char*);
```

Returns

- BAD_FUNC_ARG - A null string was passed for the return message.
- 0 - A null string was not passed for the return message.

Description

This function takes an error and converts it into human-readable text.

Remarks

String needs to be >= 80 chars.

Preconditions

None.

Example

```
char msg[80];

CRYPT_ERR_StringGet(ret, msg);
```

Parameters

Parameters	Description
int	Error code to convert.
str	Pointer to buffer to store the message. Must hold at least 80 characters.

Function

int CRYPT_ERROR_StringGet(int err, char* str)

b) Compression Functions

CRYPT_HUFFMAN_Compress Function

Compresses a block of data.

File

crypto.h

C

```
int CRYPT_HUFFMAN_Compress(unsigned char*, unsigned int, const unsigned char*, unsigned int, unsigned int);
```

Returns

- negative - error code
- positive - bytes stored in out buffer

Description

This function compresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
const unsigned char text[] = "...";
unsigned int inSz = sizeof(text);
unsigned int outSz;
unsigned char cBuffer[1024];

int ret;

ret = CRYPT_HUFFMAN_COMPRESS(cBuffer, sizeof(cBuffer), text, inSz, 0);
```

Parameters

Parameters	Description
out	Pointer to location to store the compressed data.
outSz	Maximum size of the output data in bytes.
in	Point to location of source data.
inSz	Size of the input data in bytes.
flags	Flags to control how compress operates

Function

int CRYPT_HUFFMAN_Compress(unsigned char* out, unsigned int outSz, const unsigned char* in,
unsigned int inSz, unsigned int flags)

CRYPT_HUFFMAN_DeCompress Function

Decompresses a block of data.

File

crypto.h

C

```
int CRYPT_HUFFMAN_DeCompress(unsigned char*, unsigned int, const unsigned char*, unsigned int);
```

Returns

- negative - Error code
- positive - Bytes stored in out buffer

Description

This function decompresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
unsigned char cBuffer[1024];
unsigned char dBuffer[1024];

int ret

ret = CRYPT_HUFFMAN_DeCompress(dBuffer, sizeof(dBuffer), cBuffer, msglen);
```

Parameters

Parameters	Description
out	Pointer to destination buffer
outSz	Size of destination buffer
in	Pointer to source buffer to decompress
inSz	Size of source buffer to decompress

Function

int CRYPT_HUFFMAN_DeCompress(unsigned char* out, unsigned int outSz, const unsigned char* in, unsigned int inSz)

c) MD5 Functions

CRYPT_MD5_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_MD5_DataAdd(CRYPT_MD5_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or input
- 0 - An invalid pointer was not passed to the function

Description

This function updates the hash with the data provided.

Remarks

To preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to CRYPT_MD5_DataAdd.

Preconditions

The MD5 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_MD5_DataAdd(CRYPT_MD5_CTX* md5, const unsigned char* input, unsigned int sz)

CRYPT_MD5_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_MD5_Finalize(CRYPT_MD5_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to CRYPT_MD5_DataAdd and CRYPT_MD5_Finalize.

Preconditions

The MD5 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_MD5_Finalize(CRYPT_MD5_CTX* md5, unsigned char* digest)

CRYPT_MD5_Initialize Function

Initializes the internal structures necessary for MD5 hash calculations.

File

crypto.h

C

```
int CRYPT_MD5_Initialize(CRYPT_MD5_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for MD5 hash calculations.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.

Function

int CRYPT_MD5_Initialize(CRYPT_MD5_CTX* md5)

CRYPT_MD5_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_MD5_DataSizeSet(CRYPT_MD5_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

    CRYPT_MD5_Initialize(&md5);
    CRYPT_MD5DataSizeSet(&md5, sizeof(buffer));
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_MD5_DataSizeSet(CRYPT_MD5_CTX* md5, unsigned int msgSize)

d) Random Number Generator Functions

CRYPT_RNG_BlockGenerate Function

Createa several random numbers.

File

crypto.h

C

```
int CRYPT_RNG_BlockGenerate(CRYPT_RNG_CTX*, unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function generates several random numbers and places them in the space allocated.

Preconditions

RNG context was initialized using the CRYPT_RNG_Initialize function.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to buffer to store the random numbers.
sz	Number of 8-bit random numbers to generate.

Function

int CRYPT_RNG_BlockGenerate(CRYPT_RNG_CTX* rng, unsigned char* b, unsigned int sz)

CRYPT_RNG_Get Function

Gets one random number.

File

crypto.h

C

```
int CRYPT_RNG_Get(CRYPT_RNG_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- Less than 0 - An error occurred.
- 0 or greater - Success.

Description

This function gets one random number from the random number generator.

Preconditions

RNG context was initialized using the CRYPT_RNG_Initialize function.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to 8-bit location to store the result.

Function

int CRYPT_RNG_Get(CRYPT_RNG_CTX* rng, unsigned char* b)

e) AES Encryption/Decryption Functions

CRYPT_AES_CBC_Decrypt Function

Performs AES decryption using Cipher-Block-Chaining (CBC).

File

crypto.h

C

```
int CRYPT_AES_CBC_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];
byte          out2[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CBC_Decrypt(&mcAes, out2, out1, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption pass.
in	Pointer to buffer holding the data to be decrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CBC_Decrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in, unsigned int inSz)

CRYPT_AES_CBC_Encrypt Function

Performs AES encryption using Cipher-Block-Chaining (CBC).

File

crypto.h

C

```
int CRYPT_AES_CBC_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CBC_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CBC_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in, unsigned int inSz)

CRYPT_AES_CTR_Encrypt Function

Performs AES encryption using Counter (CTR).

File

crypto.h

C

```
int CRYPT_AES_CTR_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Counter (CTR) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CTR_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CTR_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,

const unsigned char* in, unsigned int inSz)

CRYPT_AES_DIRECT_Decrypt Function

Directs decryption of one block of data.

File

crypto.h

C

```
int CRYPT_AES_DIRECT_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[CRYPT_AES_BLOCK_SIZE];
byte          out2[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Decrypt(&mcAes, out2, out1);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption.
in	Pointer to buffer where the data to decrypt is located.

Function

int CRYPT_AES_DIRECT_Decrypt(CRYPT_AES_CTX*, unsigned char*,

const unsigned char*)

CRYPT_AES_DIRECT_Encrypt Function

Directs encryption of one block of data.

File

crypto.h

C

```
int CRYPT_AES_DIRECT_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Encrypt(&mcAes, out1, ourData);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.

out	Pointer to buffer to store the results of the encryption.
in	Pointer to buffer where the data to encrypt is located.

Function

int CRYPT_AES_DIRECT_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in)

CRYPT_AES_IvSet Function

Sets the initialization vector (IV) for AES processing.

File

crypto.h

C

```
int CRYPT_AES_IvSet(CRYPT_AES_CTX*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the IV that AES will use for later processing.

Remarks

Direction, as set previously in CRYPT_AES_KeySet, is preserved.

Preconditions

The key must be set previously with CRYPT_AES_KeySet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;

strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_IvSet(&mcAes, iv);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector.

Function

int CRYPT_AES_IvSet(CRYPT_AES_CTX* aes, const unsigned char* iv)

CRYPT_AES_KeySet Function

Sets the key and initialization vector (IV) for AES processing.

File

crypto.h

C

```
int CRYPT_AES_KeySet(CRYPT_AES_CTX*, const unsigned char*, unsigned int, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and IV, and the direction (encryption or decryption) that AES will later perform.

Preconditions

None.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key itself.
keyLen	Length of key in bytes.
iv	Pointer to buffer holding the initialization vector.
dir	Which operation (CRYPT_AES_ENCRYPTION or CRYPT_AES_DECRYPTION).

Function

int CRYPT_AES_KeySet(CRYPT_AES_CTX* aes, const unsigned char* key,

unsigned int keylen, const unsigned char* iv, int dir)

f) ECC Encryption/Decryption Functions

CRYPT_ECC_DHE_KeyMake Function

Creates a new ECC key.

File

crypto.h

C

```
int CRYPT_ECC_DHE_KeyMake(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function creates a new ECC key.

Preconditions

The context must have been initialized with a call to CRYPT_ECC_Initialize. The random number generator context must have been initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
rng	Pointer to the context for the random number generator.
keySz	The size of the key desired.

Function

int CRYPT_ECC_DHE_KeyMake(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int)

CRYPT_ECC_DHE_SharedSecretMake Function

Creates an ECC shared secret between two keys.

File

crypto.h

C

```
int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX*, CRYPT_ECC_CTX*, unsigned char*, unsigned int, unsigned
int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the shared secret.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes two ECC contexts (one public, one private) and creates a shared secret between the two. The secret conforms to EC-DH from ANSI X9.63.

Preconditions

Both contexts must have been initialized with a call to CRYPT_ECC_Initialize. Both contexts have had their respective keys imported or created.

Example

```
CRYPT_ECC_CTX userA;
CRYPT_ECC_CTX userB;
int           ret;
byte          sharedA[100];
unsigned int  aSz   = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_Initialize(&userB);
...
// Make or import the appropriate keys
...
ret = CRYPT_ECC_DHE_SharedSecretMake(&userA, &userB, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
priv	Pointer to the private ECC context (with the private key).
pub	Pointer to the public ECC context (with the public key).
out	Destination of the shared secret.
outSz	The max size of the shared secret.
usedSz	Resulting size of the shared secret.

Function

int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX* priv, CRYPT_ECC_CTX* pub,

unsigned char* out, unsigned int outSz, unsigned int* usedSz)

CRYPT_ECC_DSA_HashSign Function

Signs a message digest.

File

crypto.h

C

```
int CRYPT_ECC_DSA_HashSign(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, unsigned char*, unsigned int, unsigned int*,
const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes a message digest and signs it using a private ECC key.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The RNG context must have been initialized with a call to CRYPT_RNG_Initialize. The private key used for the signature must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashSign(&userA, &mcRng, sig, sigSz, &usedA, ourData,
                             CRYPT_SHA_DIGEST_SIZE);
```

Parameters

Parameters	Description
ecc	Pointer to ECC context which saves state between calls and holds keys.
rng	Pointer to Random Number Generator context.
sig	Destination for the signature.
sigSz	The max size of the signature, in bytes.
usedSz	The resulting size of the signature, in bytes.
in	Pointer to the message digest to sign.
inSz	The length of the digest, in bytes.

Function

int CRYPT_ECC_DSA_HashSign(CRYPT_ECC_CTX* ecc, CRYPT_RNG_CTX* rng, unsigned char* sig,

unsigned int sigSz, unsigned int* usedSz, const unsigned char* in, unsigned int inSz)

CRYPT_ECC_DSA_HashVerify Function

Verifies an ECC signature.

File

crypto.h

C

```
int CRYPT_ECC_DSA_HashVerify(CRYPT_ECC_CTX*, const unsigned char*, unsigned int, unsigned char*, unsigned
int, int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the operation.
- 0 - An invalid pointer was not passed to the function.

Description

This function verifies that an ECC signature is valid.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The key used for the signature must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;
int verifyStatus = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashVerify(&userA, sig, sigSz, ourData,
                               CRYPT_SHA_DIGEST_SIZE, &verifyStatus);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
sig	The signature to verify.
sigSz	The length of the signature (octets).
hash	The hash (message digest) that was signed.
hashSz	The length of the hash (octets).
status	Result of signature (1 == valid, 0 == invalid).

Function

int CRYPT_ECC_DSA_HashVerify(CRYPT_ECC_CTX* ecc, const unsigned char* sig,

unsigned int sigSz, unsigned char* hash, unsigned int hashSz, int* status)

CRYPT_ECC_Free Function

Cleans up an Elliptic Curve Cryptography (ECC) Context.

File

crypto.h

C

```
int CRYPT_ECC_Free(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function cleans up memory allocated for an ECC Context.

Preconditions

The context must have been initialized previously with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;

ret = CRYPT_ECC_Initialize(&userA);
...
ret = CRYPT_ECC_Free(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to clean up.

Function

int CRYPT_ECC_Free(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_Initialize Function

Initializes the context for Elliptic Curve Cryptography (ECC).

File

crypto.h

C

```
int CRYPT_ECC_Initialize(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate memory for the keys.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used for Elliptic Curve Cryptography (ECC). The context is then passed to calls to perform key building, encryption, decryption, etc.

Preconditions

None.

Example

```
CRYPT_ECC_CTX userA;
int           ret;

ret = CRYPT_ECC_Initialize(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to initialize.

Function

int CRYPT_ECC_Initialize(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_KeySizeGet Function

Returns the key size in octets.

File

crypto.h

C

```
int CRYPT_ECC_KeySizeGet(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the key, in octets.

Description

This function returns the size of the ECC key, in octets.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The key must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_KeySizeGet(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls and contains the key.

Function

int CRYPT_ECC_KeySizeGet(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_PrivateImport Function

Imports private key pair in X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PrivateImport(CRYPT_ECC_CTX*, const unsigned char*, unsigned int, const unsigned char*,
unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function imports a public/private key pair in X9.63 format.

Preconditions

The context must have been initialized with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX ecc;

CRYPT_ECC_Initialize(&ecc);
...
CRYPT_ECC_PrivateImport(&ecc, priv_key, sizeof(priv_key), pub_key, sizeof(pub_key));
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.

priv	Pointer to the private key.
privSz	Size of the private key, in bytes.
pub	Pointer to the public key.
pubSz	Size of the public key, in bytes.

Function

int CRYPT_ECC_PrivateImport(CRYPT_ECC_CTX* ecc, const unsigned char* priv,
unsigned int privSz, const unsigned char* pub,
unsigned int pubSz)

CRYPT_ECC_PublicExport Function

Exports public ECC key in ANSI X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PublicExport(CRYPT_ECC_CTX*, unsigned char*, unsigned int, unsigned int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- BUFFER_E - The output buffer was too small to store the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes an ECC public key and exports it in ANSI X9.63 format.

Preconditions

The context must be initialized previously with a call to CRYPT_ECC_Initialize. The key must also have been constructed with a call to CRYPT_ECC_DHE_KeyMake. A random number generator must all have been initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);
ret = CRYPT_ECC_PublicExport(&userA, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
out	Buffer in which to store the public key.
outSz	The available size of the buffer, in bytes.
usedSz	Return value indicating how many bytes were used.

Function

int CRYPT_ECC_PublicExport(CRYPT_ECC_CTX* ecc, unsigned char* out,
unsigned int outSz, unsigned int* usedSz)

CRYPT_ECC_PublicImport Function

Imports public key in ANSI X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PublicImport(CRYPT_ECC_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the key.
- ASN_PARSE_E - There was a parse error while going through the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function imports a public key in ANSI X9.63 format into the ECC context.

Preconditions

The ECC context must have previously been initialized with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX userB;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA;

ret = CRYPT_ECC_Initialize(&userB);
...
ret = CRYPT_ECC_PublicImport(&userB, sharedA, usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
in	Input buffer the holds the public key.
inSz	Size of the input buffer, in bytes.

Function

int CRYPT_ECC_PublicImport(CRYPT_ECC_CTX* ecc, const unsigned char* in,

unsigned int inSz)

CRYPT_ECC_SignatureSizeGet Function

Returns the signature size in octets.

File

crypto.h

C

```
int CRYPT_ECC_SignatureSizeGet(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the signature.

Description

This function returns the size of the signature in a given context, in octets.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The keys must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
```

```
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_SignatureSizeGet(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls, and contains the signature.

Function

int CRYPT_ECC_SignatureSizeGet(CRYPT_ECC_CTX* ecc)

g) RSA Encryption/Decryption Functions

CRYPT_RSA_EncryptSizeGet Function

Gets the size of the RSA Key.

File

crypto.h

C

```
int CRYPT_RSA_EncryptSizeGet(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the key.

Description

This function retrieves the size of the RSA Key in use for the context.

Preconditions

The context must be initialized with a call to CRYPT_RSA_Initialize and the keys decoded either with CRYPT_RSA_PrivateKeyDecode or CRYPT_RSA_PublicKeyDecode.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_EncryptSizeGet(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

int CRYPT_RSA_EncryptSizeGet(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_Free Function

Releases the memory used for the key and clean up the context.

File

crypto.h

C

```
int CRYPT_RSA_Free(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function releases the memory used during RSA processing for storing the public/private key.

Preconditions

The context must have been set up previously with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;

ret = CRYPT_RSA_Initialize(&mcRsa);
ret = CRYPT_RSA_Free(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

int CRYPT_RSA_Free(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_Initialize Function

Initializes the internal structures necessary for RSA processing.

File

crypto.h

C

```
int CRYPT_RSA_Initialize(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate the memory necessary for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used during public-key RSA encryption and decryption.

Preconditions

None.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;

ret = CRYPT_RSA_Initialize(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to RSA context which saves state between calls.

Function

int CRYPT_RSA_Initialize(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_PrivateDecrypt Function

Decrypts data using a private key.

File

crypto.h

C

```
int CRYPT_RSA_PrivateDecrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned char*, unsigned
int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function decrypts a data block using a private key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Private Key Decoded using CRYPT_RSA_PrivateKeyDecode prior to calling this function.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_PrivateDecrypt(&mcRsa, out2, sizeof(out2), out1,
                               RSA_TEST_SIZE);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be decrypted.
inSz	Size of input buffer.

Function

int CRYPT_RSA_PrivateDecrypt(CRYPT_RSA_CTX* rsa, unsigned char* out,

unsigned int outSz, const unsigned char* in, unsigned int inSz)

CRYPT_RSA_PrivateKeyDecode Function

Constructs the Private Key from a DER certificate.

File

crypto.h

C

```
int CRYPT_RSA_PrivateKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a private key from a DER-formated certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the private key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PrivateKeyDecode(CRYPT_RSA_CTX*, const unsigned char*,

unsigned int)

CRYPT_RSA_PublicEncrypt Function

Encrypts data using a public key.

File

crypto.h

C

```
int CRYPT_RSA_PublicEncrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned char*, unsigned
int, CRYPT_RNG_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function encrypts a data block using a public key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Public Key Decoded using CRYPT_RSA_PublicKeyDecode prior to calling this function. The random number generator must be initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
CRYPT_RNG_CTX mcRng;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);
ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

```
ret = CRYPT_RSA_PublicEncrypt(&mcRsa, out1, sizeof(out1), ourData,
                              RSA_TEST_SIZE, &mcRng);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be encrypted.
inSz	Size of input buffer.
rng	Pointer to Random Number Generator (RNG) context.

Function

int CRYPT_RSA_PublicEncrypt(CRYPT_RSA_CTX* rsa, unsigned char* out,
unsigned int outSz, const unsigned char* in, unsigned int inSz,
CRYPT_RNG_CTX* rng)

CRYPT_RSA_PublicKeyDecode Function

Constructs the Public Key from a DER certificate.

File

crypto.h

C

```
int CRYPT_RSA_PublicKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a public key from a DER-formated certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the public key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PublicKeyDecode(CRYPT_RSA_CTX* rsa, const unsigned char* in,
unsigned int inSz)

h) Triple DES (3DES) Encryption/Decryption Functions

CRYPT_TDES_CBC_Decrypt Function

Decrypts a data block using Triple DES.

File

crypto.h

C

```
int CRYPT_TDES_CBC_Decrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using a Triple DES algorithm.

Remarks

Input data must have a length a multiple of 8 bytes. Output data will be zero-padded at the end if the original data was not a multiple of 8 bytes long.

Preconditions

The context tdes must be set earlier using CRYPT_TDES_KeySet. The input block must be a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

int CRYPT_TDES_CBC_Decrypt(CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)

CRYPT_TDES_CBC_Encrypt Function

Encrypts a data block using Triple DES.

File

crypto.h

C

```
int CRYPT_TDES_CBC_Encrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using a Triple DES algorithm.

Remarks

The input data must be padded at the end with zeros to make the length a multiple of 8.

Preconditions

The context tdes must be set earlier using CRYPT_TDES_KeySet. The input block must be a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

int CRYPT_TDES_CBC_Encrypt(CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)

CRYPT_TDES_IvSet Function

Sets the Initialization Vector (IV) for a Triple DES operation.

File

crypto.h

C

```
int CRYPT_TDES_IvSet(CRYPT_TDES_CTX*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function changes the IV of a TDES context, but leaves the Key alone.

Remarks

The IV must be 8 bytes long.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_IvSet(&mcDes3, iv);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.

Function

int CRYPT_TDES_IvSet(CRYPT_TDES_CTX* tdes, const unsigned char* iv)

CRYPT_TDES_KeySet Function

Initialization of Triple DES context.

File

crypto.h

C

```
int CRYPT_TDES_KeySet(CRYPT_TDES_CTX*, const unsigned char*, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and initialization vector (IV) for a set of Triple-DES operations.

Remarks

The input data must be a multiple of 8 bytes, and must be padded at the end with zeros to meet the length.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);
```

```
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key. Must be 24 bytes in size.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.
dir	Indicates whether encryption or decryption will be done. Can be set to: • CRYPT_TDES_ENCRYPTION - For Encryption operations • CRYPT_TDES_DECRYPTION - Fro Decryption operations

Function

int CRYPT_TDES_KeySet(CRYPT_TDES_CTX* tdes, const unsigned char* key, const unsigned char* iv, int dir)

i) HMAC Hash Functions

CRYPT_HMAC_DataAdd Function

Adds data to the HMAC calculation.

File

crypto.h

C

```
int CRYPT_HMAC_DataAdd(CRYPT_HMAC_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function adds data to the HMAC so that multiple blocks of data can be processed.

Remarks

None.

Preconditions

The CRYPT_HMAC_CTX context must be initialized using the CRYPT_HMAC_SetKey function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context that saves state between calls.
sz	Size of the input data in bytes.

Function

int CRYPT_HMAC_DataAdd(CRYPT_HMAC_CTX*, const unsigned char*, unsigned int)

CRYPT_HMAC_Finalize Function

Completes the HMAC calculation and get the results.

File

crypto.h

C

```
int CRYPT_HMAC_Finalize(CRYPT_HMAC_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function completes the HMAC calculations. The results are placed in the location pointed to by digest.

Remarks

The area pointed to by digest must be large enough to hold the results.

Preconditions

The CRYPT_HMAC_CTX context must be initialized using the CRYPT_HMAC_SetKey function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
digest	Pointer to place to put the final HMAC digest results.

Function

int CRYPT_HMAC_Finalize(CRYPT_HMAC_CTX* hmac, unsigned char* digest)

CRYPT_HMAC_SetKey Function

Initializes the HMAC context and set the key for the hash.

File

crypto.h

C

```
int CRYPT_HMAC_SetKey(CRYPT_HMAC_CTX*, int, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the HMAC context and set the key for the hash.

Remarks

None.

Preconditions

None.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
type	Type of SHA operation to use with HMAC. Must be one of the
following	• CRYPT_HMAC_SHA • CRYPT_HMAC_SHA256 • CRYPT_HMAC_SHA384 • CRYPT_HMAC_SHA512
key	Secret key used for the hash.
sz	Size of the input data in bytes.

Function

int CRYPT_HMAC_SetKey(CRYPT_HMAC_CTX* hmac, int type, const unsigned char* key, unsigned int sz)

j) SHA Hash functions

CRYPT_SHA_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA_DataAdd(CRYPT_SHA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd.

Preconditions

The SHA context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
```

```
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA_DataAdd(CRYPT_SHA_CTX* sha, const unsigned char* input, unsigned int sz)

CRYPT_SHA_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA_Finalize(CRYPT_SHA_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd and CRYPT_SHA_Finalize.

Preconditions

The SHA context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA_Finalize(CRYPT_SHA_CTX* sha, unsigned char* digest)

CRYPT_SHA_Initialize Function

Initializes the internal structures necessary for SHA hash calculations.

File

crypto.h

C

```
int CRYPT_SHA_Initialize(CRYPT_SHA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.

Function

int CRYPT_SHA_Initialize(CRYPT_SHA_CTX* sha)

CRYPT_SHA_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_SHA_DataSizeSet(CRYPT_SHA_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shasum[SHA_DIGEST_SIZE];

    CRYPT_SHA_Initialize(&sha);
    CRYPT_SHADataSizeSet(&sha, sizeof(buffer));
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shasum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_SHA_DataSizeSet(CRYPT_SHA_CTX* sha, unsigned int msgSize)

k) SHA256 Hash Functions

CRYPT_SHA256_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA256_DataAdd(CRYPT_SHA256_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha256 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to CRYPT_SHA256_DataAdd.

Preconditions

The SHA256 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA256_DataAdd(CRYPT_SHA256_CTX* sha256, const unsigned char* input, unsigned int sz)

CRYPT_SHA256_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA256_Finalize(CRYPT_SHA256_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to CRYPT_SHA256_DataAdd and CRYPT_SHA256_Finalize.

Preconditions

The SHA256 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA256_Finalize(CRYPT_SHA256_CTX* sha256, unsigned char* digest)

CRYPT_SHA256_Initialize Function

Initializes the internal structures necessary for SHA256 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA256_Initialize(CRYPT_SHA256_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA256 hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha;
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha);
CRYPT_SHA256_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to context which saves state between calls.

Function

int CRYPT_SHA256_Initialize(CRYPT_SHA256_CTX* sha256)

CRYPT_SHA256_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_SHA256_DataSizeSet(CRYPT_SHA256_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA256 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t sha256sum[SHA256_DIGEST_SIZE];

    CRYPT_SHA256_Initialize(&sha256);
    CRYPT_SHA256DataSizeSet(&sha256, sizeof(buffer));
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, sha256sum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_SHA256_DataSizeSet(CRYPT_SHA256_CTX* sha256, unsigned int msgSize)

l) SHA384 Hash Functions

CRYPT_SHA384_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA384_DataAdd(CRYPT_SHA384_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to CRYPT_SHA384_DataAdd.

Preconditions

The SHA384 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA384_DataAdd(CRYPT_SHA384_CTX* sha384, const unsigned char* input, unsigned int sz)

CRYPT_SHA384_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA384_Finalize(CRYPT_SHA384_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to CRYPT_SHA384_DataAdd and CRYPT_SHA384_Finalize.

Preconditions

The SHA384 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA384_Finalize(CRYPT_SHA384_CTX* sha384, unsigned char* digest)

CRYPT_SHA384_Initialize Function

Initializes the internal structures necessary for SHA384 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA384_Initialize(CRYPT_SHA384_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function
- 0 - An invalid pointer was not passed to the function

Description

This function initializes the internal structures necessary for SHA384 hash calculations.

Remarks

All SHA384 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.

Function

int CRYPT_SHA384_Initialize(CRYPT_SHA384_CTX* sha384)

m) SHA512 Hash Functions

CRYPT_SHA512_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA512_DataAdd(CRYPT_SHA512_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to CRYPT_SHA512_DataAdd.

Preconditions

The SHA512 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA512_DataAdd(CRYPT_SHA512_CTX* sha512, const unsigned char* input, unsigned int sz)

CRYPT_SHA512_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA512_Finalize(CRYPT_SHA512_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to CRYPT_SHA512_DataAdd and CRYPT_SHA512_Finalize.

Preconditions

The SHA512 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA512_Finalize(CRYPT_SHA512_CTX* sha512, unsigned char* digest)

CRYPT_SHA512_Initialize Function

Initializes the internal structures necessary for SHA512 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA512_Initialize(CRYPT_SHA512_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA512 hash calculations.

Remarks

All SHA512 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.

Function

int CRYPT_SHA512_Initialize(CRYPT_SHA512_CTX* sha512)

n) Data Types and Constants

CRYPT_AES_CTX Structure

File

crypto.h

C

```
struct CRYPT_AES_CTX {
  int holder[90];
};
```

Members

Members	Description
int holder[90];	big enough to hold internal, but check on init

Description

AES

CRYPT_ECC_CTX Structure

File

crypto.h

C

```
struct CRYPT_ECC_CTX {
  void* holder;
};
```

Description

ECC

CRYPT_HMAC_CTX Structure

File

crypto.h

C

```
struct CRYPT_HMAC_CTX {
  long long holder[80];
};
```

Members

Members	Description
long long holder[80];	big enough to hold internal, but check on init

Description

HMAC

CRYPT_MD5_CTX Structure

File

crypto.h

C

```
struct CRYPT_MD5_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

MD5

CRYPT_RNG_CTX Structure

File

crypto.h

C

```
struct CRYPT_RNG_CTX {
  int holder[66];
};
```

Members

Members	Description
int holder[66];	big enough to hold internal, but check on init

Description

RNG

CRYPT_RSA_CTX Structure

File

crypto.h

C

```
struct CRYPT_RSA_CTX {
  void* holder;
};
```

Description

RSA

CRYPT_SHA_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

SHA

CRYPT_SHA256_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA256_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

SHA-256

CRYPT_SHA384_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA384_CTX {
  long long holder[32];
};
```

Members

Members	Description
long long holder[32];	big enough to hold internal, but check on init

Description

SHA-384

CRYPT_SHA512_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA512_CTX {
  long long holder[36];
};
```

Members

Members	Description
long long holder[36];	big enough to hold internal, but check on init

Description

SHA-512

CRYPT_TDES_CTX Structure

File

crypto.h

C

```
struct CRYPT_TDES_CTX {
  int holder[104];
};
```

Members

Members	Description
int holder[104];	big enough to hold internal, but check on init

Description

TDES

MC_CRYPTO_API_H Macro

File

crypto.h

C

```
#define MC_CRYPTO_API_H
```

Description

Defines Microchip CRYPTO API layer

Files

Files

Name	Description
crypto.h	Crypto Framework Library header for cryptographic functions.

Description

This section lists the source and header files used by the Crypto Library.

crypto.h

Crypto Framework Library header for cryptographic functions.

Functions

	Name	Description
	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.
	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.
	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
	CRYPT_ECC_DSA_HashSign	Signs a message digest.
	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
	CRYPT_ECC_KeySizeGet	Returns the key size in octets.
	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.
	CRYPT_ERROR_StringGet	Reports the nature of an error.
	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.
	CRYPT_HUFFMAN_Compress	Compresses a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
	CRYPT_RNG_BlockGenerate	Createa several random numbers.
	CRYPT_RNG_Get	Gets one random number.
	CRYPT_RNG_Initialize	Initializes the random number generator.
	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
	CRYPT_RSA_Free	Releases the memory used for the key and clean up the context.
	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.

	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

Macros

	Name	Description
	MC_CRYPTO_API_H	Defines Microchip CRYPTO API layer

Structures

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES

Description

Crypto Framework Library Header

This header file contains function prototypes and definitions of the data types and constants that make up the Cryptographic Framework Library for PIC32 families of Microchip microcontrollers.

File Name

crypto.h

Company

Microchip Technology Inc.

Decoder Libraries Help

This section provides descriptions of the software Decoder libraries that are available in MPLAB Harmony.

AAC Decoder Library

This section describes the AAC Decoder Library.

Introduction

Advanced Audio Coding (AAC) is a proprietary audio coding standard for lossy digital audio compression. Designed to be the successor of the MP3 format, AAC generally achieves better sound quality than MP3 at the same bit rate. The AAC Decoder Library in MPLAB Harmony is a decoder for Low-Complexity Profile (AAC-LC), which is available for the PIC32MX and PIC32MZ family of microcontrollers.

Description

The AAC Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

This AAC algorithm Non-Modifiable Binary Code is designed for 80 MHz or greater PIC32MX MCUs. Keep in mind, this code requires 62 MIPS peak 34 MIPS average performance, 61 KB Flash and 12 KB RAM without frame buffer memory for operation on the PIC32MX MCU. This product is the non-modifiable binary. A modifiable source-code version of this code is also available on Microchip Direct (SW320013-2). Users are responsible for licensing for their products through Via Licensing.

Features

- Supports the MPEG-4 AAC-LC Profile
- Supports Audio Data Transport Stream (ADTS) input formats
- Supports Sampling Rates from 8 kHz to 96 kHz
- Supports bitrates from 8 kbps to 1152 kbps

Library Performance

The following matrix was measured on a PIC32MX family device.

	Performance Statistics (Average)	Memory Statistics	
AAC 44.1 kHz Test Vector	MIPS	Data RAM	Flash Memory
	73	26.72 KB	47.13 KB

Input buffer for one frame: 1536 bytes

Output Buffer: 4096 bytes for 16-bit audio sample, stereo output

Using the Library

This topic describes the basic architecture of the AAC Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_aac.h`

The interface to the AAC Decoder Library is defined in the `decoder_aac.h` header file. Any C language source (`.c`) file that uses the AAC Decoder Library should include `decoder_aac.h`.

Please refer to the What is MPLAB Harmony? section for how the AAC Decoder Library interacts with the framework.

Please refer to Universal Audio Decoders (universal_audio_decoders) demonstration to see how to use the MPLAB Harmony AAC framework APIs to decode a AAC frame.

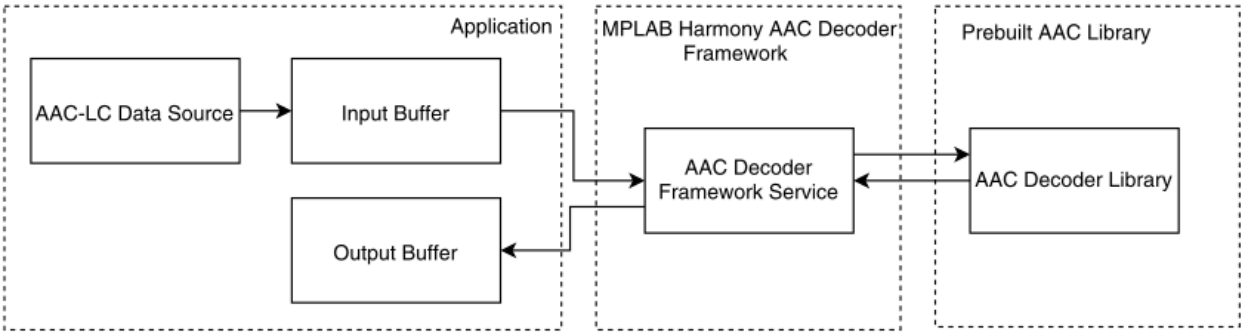
Abstraction Model

This library provides an abstraction of the MPEG4 AAC-LC Decoder.

Description

The following block diagram describes how the AAC Decoder Library interacts with an application. The AAC Decoder Framework Service is another layer of abstraction of the AAC decoder library, which provides application APIs to set and get decoder state information.

Abstraction Model



Library Overview

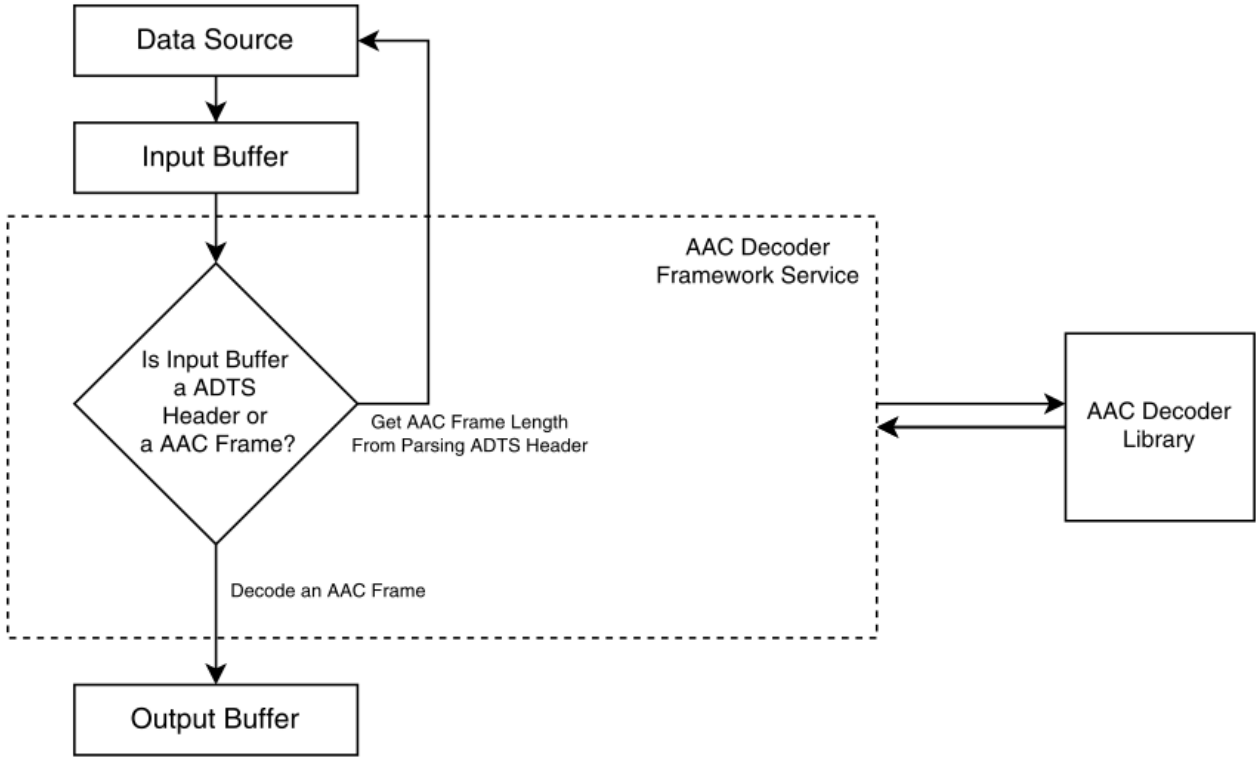
The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the AAC Decoder Library module.

How the Library Works

This topic provides information on how the library works.

Description

The following diagram describes the data flow between the AAC Decoder and an application.



Initializing the AAC Decoder

Initialize the AAC Decoder by calling function AAC_Initialize from the `aac_dec.h` file. This function initializes the AAC decoder state structure, and obtains streaming information from the first ADTS header. The application can then retrieve stream information by using the APIs in `aac_dec.h` after initialization.

Decoding a AAC Frame

The AAC_Decoder function in `aac_dec.h` is used to decode a single AAC frame, and depending on the current state of the AAC Decoder, this function either parses a ADTS header to get the next AAC frame size, or decodes a single AAC frame.

Code Example

```
void DECODER_Initialize(decoder_type){
       switch(decoder_type){
```

```
                case AAC_DECODER:
                    bool ret = AAC_Initialize(aacDecoderPtr,sizeOfAACDecoder, adtsPtr, aacFilehandle)
                    if(ret)
                    {
                        // setup other components,
                        // for example, audio CODEC, display, etc...
                    }
                }
        }
void App_Task()
{
        while(1){
                while(not end of audio data){
                    // Read aac frame to input buffer pointer
                    appData.nBytesRead = OPUS_DiskRead(input_ptr);
                    // Decode aac frame
                    bool ret = AAC_Decoder (input_ptr,inSize,read,output,written,outBufSize);
                    if(ret)
                    {
                        // continue decoding
                    }else{
                        // handle decoding errors;
                    }
                }
                break;
        }
}
```

Library Interface

a) General Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.
	AAC_GetChannels	This is function AAC_GetChannels.

b) Data Types and Constants

	Name	Description
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.
	SetReadBytesInAppData	This is type SetReadBytesInAppData.
	AAC_DEC_H	This is macro AAC_DEC_H.

Description

This section describes the Application Programming Interface (API) functions of the AAC Decoder Library.

Refer to each section for a detailed description.

a) General Functions

AAC_Decoder Function

File

aac_dec.h

C

```
int16_t AAC_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
```

```
written);
```

Description

This is function AAC_Decoder.

isAACdecoder_enabled Function

File

aac_dec.h

C

```
bool isAACdecoder_enabled();
```

Description

This is function isAACdecoder_enabled.

AAC_Initialize Function

File

aac_dec.h

C

```
bool AAC_Initialize(void * heap, uint16_t size, uint8_t * ptr, SYS_FS_HANDLE aacFilehandle);
```

Description

This is function AAC_Initialize.

AAC_RegisterDecoderEventHandlerCallback Function

File

aac_dec.h

C

```
void AAC_RegisterDecoderEventHandlerCallback(SetReadBytesInAppData fptr);
```

Description

This is function AAC_RegisterDecoderEventHandlerCallback.

AAC_GetSamplingFrequency Function

File

aac_dec.h

C

```
int32_t AAC_GetSamplingFrequency(uint8_t * ptr);
```

Description

This is function AAC_GetSamplingFrequency.

AAC_GetChannels Function

File

aac_dec.h

C

```
uint8_t AAC_GetChannels();
```

Description

This is function AAC_GetChannels.

b) Data Types and Constants

AAC_DECODER_STATES Enumeration

File

aac_dec.h

C

```
typedef enum {
  AAC_GET_FRAME_SIZE,
  AAC_DECODE_FRAME
} AAC_DECODER_STATES;
```

Description

This is type AAC_DECODER_STATES.

AAC_SAMPLING_FREQUENCY_INDEX Enumeration

File

aac_dec.h

C

```
typedef enum {
  SAMPLING_FREQUENCY_IDX0 = 0,
  SAMPLING_FREQUENCY_IDX1,
  SAMPLING_FREQUENCY_IDX2,
  SAMPLING_FREQUENCY_IDX3,
  SAMPLING_FREQUENCY_IDX4,
  SAMPLING_FREQUENCY_IDX5,
  SAMPLING_FREQUENCY_IDX6,
  SAMPLING_FREQUENCY_IDX7,
  SAMPLING_FREQUENCY_IDX8,
  SAMPLING_FREQUENCY_IDX9,
  SAMPLING_FREQUENCY_IDX10,
  SAMPLING_FREQUENCY_IDX11,
  SAMPLING_FREQUENCY_IDX12,
  SAMPLING_FREQUENCY_IDX13,
  SAMPLING_FREQUENCY_IDX14,
  SAMPLING_FREQUENCY_IDX15
} AAC_SAMPLING_FREQUENCY_INDEX;
```

Description

This is type AAC_SAMPLING_FREQUENCY_INDEX.

AAC_ERROR_COUNT_MAX Macro

File

aac_dec.h

C

```
#define AAC_ERROR_COUNT_MAX 1
```

Description

This is macro AAC_ERROR_COUNT_MAX.

SetReadBytesInAppData Type

File

aac_dec.h

C

```
typedef void (* SetReadBytesInAppData)(int32_t val);
```

Description

This is type SetReadBytesInAppData.

AAC_DEC_H Macro

File

aac_dec.h

C

```
#define AAC_DEC_H
```

Description

This is macro AAC_DEC_H.

Files

Files

Name	Description
aac_dec.h	AAC decoder interface header file.

Description

This section lists the source and header files used by the MP3 Decoder Library.

aac_dec.h

AAC decoder interface header file.

Enumerations

	Name	Description
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.

Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	AAC_GetChannels	This is function AAC_GetChannels.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.

Macros

	Name	Description
	AAC_DEC_H	This is macro AAC_DEC_H.
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.

Types

	Name	Description
	SetReadBytesInAppData	This is type SetReadBytesInAppData.

Description

AAC Decoder Library Support File

This header file describes the interface functions of AAC decoder.

File Name

aac_dec.h

Company

Microchip Technology Inc.

FLAC Decoder Library

This section describes the FLAC Decoder Library.

Introduction

FLAC stands for Free Lossless Audio Codec, which is an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. This Library is Based on xiph.org's FLAC 1.3.1 source code, and is under xiph.org's BSD license. This source code of FLAC 1.3.1 is modified to use input and output buffer from application instead of dynamic allocating in library.

Description

FLAC (Free Lossless Audio Codec) is an audio coding format for lossless compression of digital audio, and is also the name of the reference codec implementation. Digital audio compressed by the FLAC algorithm can typically be reduced to 50–60% of its original size and decompresses to an identical copy of the original audio data.

Features

- Lossless decoder
- Supports native FLAC format streaming
- No theoretical limitation on maximum Sample rate and bitrate

Note: Lossless decoding will consume large buffer size in runtime, therefore, refer to the FLAC performance table to determine whether it fits in your application.

Library Performance

The following matrix was measured on a PIC32MX family device.

FLAC Type	Performance Statistics (Average)	Memory Statistics	
	MIPS	Data RAM	Flash Memory
96kHz_24bits Native FLAC	54	32 KB (always the same size with output buffer)	113 KB
44.1kHz_24bits Native FLAC	42	9K	113 KB

Input buffer for one FLAC Frame: 16 KB
(to be practical it should be at least 1 KB, 16 KB works for most native FLAC frames)
Output Buffer: at least 32 KB + 32 bytes
(padding bytes because of FLAC algorithm for a 4096 samples FLAC frame)

Using the Library

This topic describes the basic architecture of the FLAC Decoder Library and provides information and examples on its use.

Description

Interface Header File: `flac.h`

The interface to the FLAC Decoder Library is defined in the `flac.h` header file. Any C language source (`.c`) file that uses the FLAC Decoder Library should include `flac.h`.

Please refer to the What is MPLAB Harmony? section for how the FLAC Decoder Library interacts with the framework.

Please refer to Universal Audio Decoders (univeral_audio_decoders) demonstration to see how to use the MPLAB Harmony FLAC framework APIs to decode a FLAC Frame.

Abstraction Model

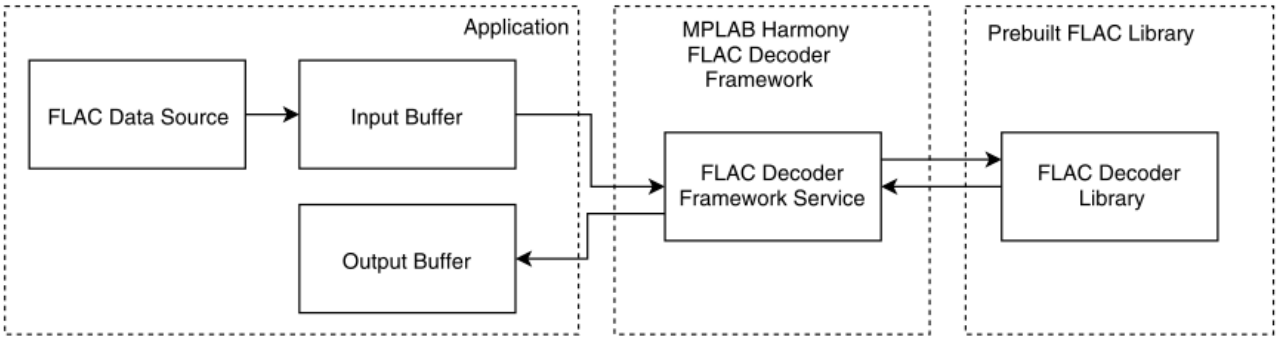
Describes the abstraction model for the FLAC Decoder Library.

Description

The following block diagram describes how the FLAC decoder library interacts with application.

The FLAC Decoder Framework Service is another layer of abstraction of the FLAC Decoder Library, which provides application APIs to set and get decoder state information.

Abstraction Model



Library Overview

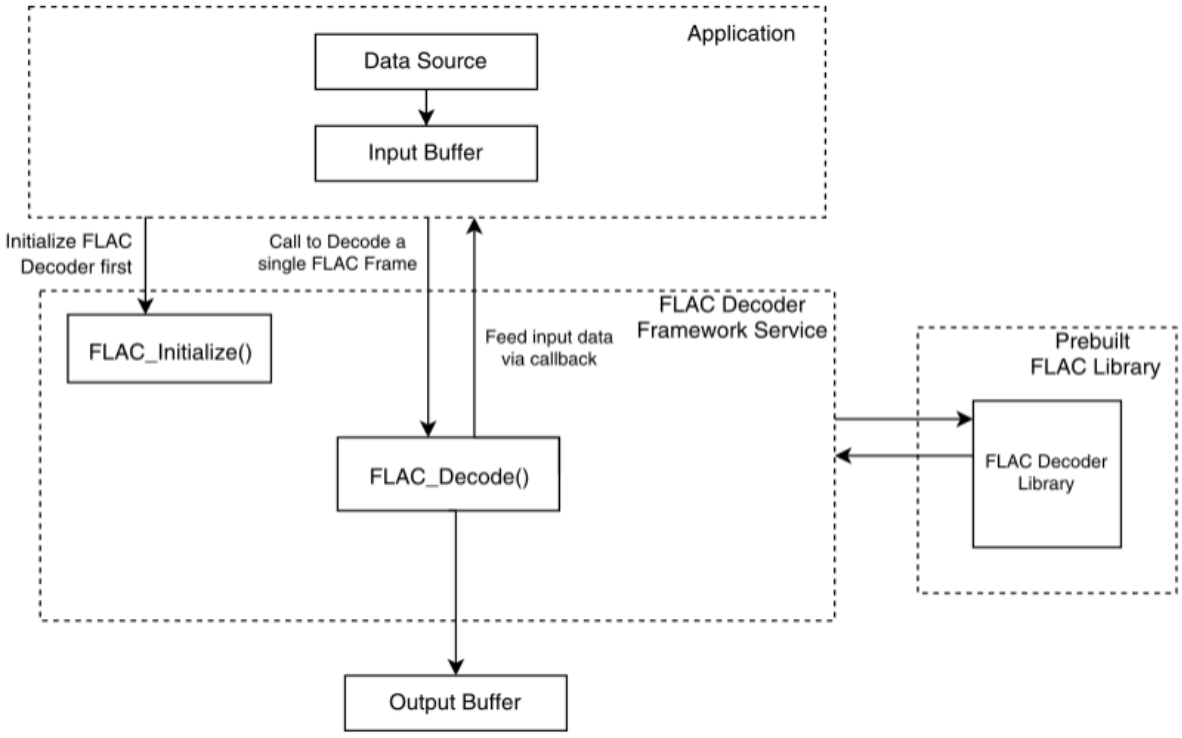
The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the FLAC Decoder Library module.

How the Library Works

This section describes how to work with the MPLAB Harmony FLAC abstraction layer interface, which provides simplifier APIs for using the FLAC Decoder Library.

Description

The following diagram describes the data flow between the FLAC Decoder and an application.



Initializing the FLAC Decoder

Initialize the FLAC Decoder by calling the function FLAC_Initialize from the `flac_dec.h` file. This function initializes the FLAC Decoder state structure, and sets callback functions for reading FLAC data. The application can start to decode FLAC frame after initialization.

Decoding a FLAC Frame

The FLAC_Decoder function in `flac_dec.h` is used to decode a single FLAC frame, and write back decoded data in an output buffer.

Code Example

This FLAC decoder framework service provides general APIs for decoder application use. The following code example shows typical usage of this FLAC framework service.

```
[Code Example]

void DECODER_Initialize(decoder_type){
        switch(decoder_type){
            case FLAC:
       if(FLAC_Initialize (appDataPtr->fileHandle, appDataPtr->fileStatus.lfname) == true)
       {
        // setup other components,
        // for example, audio CODEC, display, etc...
          }
        }
}

void App_Task()
{
         while(audio is not end){
    // Read one Opus packet
    appData.nBytesRead = SYS_File_Read(file_ptr, input_ptr, blocksize);
    // Decode one packet
           ret = FLAC_Decoder (input_ptr,inSize,read,output,written);
    if(ret == success)
    {
         // continue decoding
    }else{
        // handle decoding errors;
    }
         }
         // Clean up
         FLAC_Cleanup();
     }
}
```

Library Interface

a) Functions

	Name	Description
	FLAC_Cleanup	A clean up function for deallocating memory resources of a FLAC decoder.
	FLAC_Decoder	An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.
	FLAC_GetBitRate	Returns bit rate of the FLAC audio file.
	FLAC_GetBlockSize	Returns size of next packet to be decoded.
	FLAC_GetChannels	Returns number of channel of the FLAC file.
	FLAC_GetSamplingRate	Returns sample rate of the FLAC audio file.
	FLAC_RegisterDecoderEventHandlerCallback	Register a decoder event handler function to FLAC decoder.
	isFLACdecoder_enabled	Return a boolean if the FLAC decoder is included or not.
	FLAC_GetBitdepth	Returns bitdepth of the FLAC audio file.
	FLAC_GetDuration	Returns track length of the FLAC audio file.
	FLAC_Initialize	An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.

b) Data Types and Constants

	Name	Description
	FLAC_DEC_H	This is macro FLAC_DEC_H.

Description

This section describes the Application Programming Interface (API) functions of the FLAC Decoder Library.

Refer to each section for a detailed description.

a) Functions

FLAC_Cleanup Function

A clean up function for deallocating memory resources of a FLAC decoder.

File

flac_dec.h

C

```
void FLAC_Cleanup();
```

Returns

Size of next packet to be decoded.

Description

Function FLAC_Cleanup:

void FLAC_Cleanup();

This function must be called after a FLAC audio file is decoded to free the memory resources.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
blocksize = FLAC_GetBlockSize();
SYS_FS_FileRead(fptr, input, blocksize);
```

FLAC_Decoder Function

An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.

File

flac_dec.h

C

```
bool FLAC_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, uint8_t * output, uint16_t * written);
```

Returns

output buffer pointer which holds decoded data. written - size of decoded data.

This function returns a boolean value.

- 0 - FLAC decoder decodes failed.
- 1 - FLAC decoder decodes succeed.

Description

Function FLAC_Decoder:

bool FLAC_Decoder(uint8_t *input, uint16_t inSize, uint16_t *read, uint8_t *output, uint16_t *written);

This function decodes input buffer then returns decoded data, it provides a abstraction interface to use FLAC decode functions from library.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
SYS_FS_FileRead(fptr, input, inSize);
FLAC_Decoder(input, inSize, read, output, written);
```

Parameters

Parameters	Description
inSize	input buffer size
read	size of input buffer that is read by this function.

FLAC_GetBitRate Function

Returns bit rate of the FLAC audio file.

File

flac_dec.h

C

```
int32_t FLAC_GetBitRate();
```

Returns

Bit rate of the FLAC audio file.

Description

Function FLAC_GetBitRate:

int32_t FLAC_GetBitRate();

This function returns bit rate of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
bitrate = FLAC_GetBitRate();
```

FLAC_GetBlockSize Function

Returns size of next packet to be decoded.

File

flac_dec.h

C

```
int32_t FLAC_GetBlockSize();
```

Returns

Size of next packet to be decoded.

Description

Function FLAC_GetBlockSize:

int32_t FLAC_GetBlockSize();

This function returns size of next packet to be decoded.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
blocksize = FLAC_GetBlockSize();
SYS_FS_FileRead(fptr, input, blocksize);
```

FLAC_GetChannels Function

Returns number of channel of the FLAC file.

File

flac_dec.h

C

```
uint8_t FLAC_GetChannels();
```

Returns

Number of audio channels

Description

Function FLAC_GetChannels:

uint8_t FLAC_GetChannels();

This function returns audio channel number of the FLAC file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
num_channel = FLAC_GetChannels();
```

FLAC_GetSamplingRate Function

Returns sample rate of the FLAC audio file.

File

flac_dec.h

C

```
int32_t FLAC_GetSamplingRate();
```

Returns

Sample rate of the FLAC audio file.

Description

Function FLAC_GetSamplingRate:

int32_t FLAC_GetSamplingRate();

This function returns sample rate of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
samplerate = FLAC_GetSamplingRate();
```

FLAC_RegisterDecoderEventHandlerCallback Function

Register a decoder event handler function to FLAC decoder.

File

flac_dec.h

C

```
void FLAC_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);
```

Returns

None.

Description

Function FLAC_RegisterDecoderEventHandlerCallback:

void FLAC_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);

This function registers a event handler function for propagating FLAC decoding information to a upper level.

Remarks

None.

Preconditions

None.

Example

```
FLAC_RegisterDecoderEventHandlerCallback(fptr);
```

Parameters

Parameters	Description
fptr	event handler function pointer.

isFLACdecoder_enabled Function

Return a boolean if the FLAC decoder is included or not.

File

flac_dec.h

C

```
bool isFLACdecoder_enabled();
```

Returns

This function returns a boolean value.

- 0 - FLAC decoder is not enabled.
- 1 - FLAC decoder is enabled.

Description

Function isFLACdecoder_enabled:

bool isFLACdecoder_enabled();

Return a boolean if the FLAC decoder is included or not.

Remarks

None.

Preconditions

None.

Example

```
bool flac_enabled = isFLACdecoder_enabled();
```

FLAC_GetBitdepth Function

Returns bitdepth of the FLAC audio file.

File

flac_dec.h

C

```
uint8_t FLAC_GetBitdepth();
```

Returns

Bitdepth of the FLAC audio file.

Description

Function FLAC_GetBitdepth:

uint8_t FLAC_GetBitdepth();

This function returns bitdepth of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
bitdepth = FLAC_GetBitdepth();
```

FLAC_GetDuration Function

Returns track length of the FLAC audio file.

File

flac_dec.h

C

```
uint32_t FLAC_GetDuration();
```

Returns

track length of the FLAC audio file.

Description

Function FLAC_GetDuration:

uint32_t FLAC_GetDuration();

This function returns track length of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
length_in_seconds = FLAC_GetDuration();
```

FLAC_Initialize Function

An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.

File

flac_dec.h

C

```
bool FLAC_Initialize(SYS_FS_HANDLE fhandle, void * input_buffer);
```

Returns

This function returns a boolean value.

- 0 - FLAC decoder initialization failed.
- 1 - FLAC decoder initialization succeed.

Description

Function FLAC_Initialize:

bool FLAC_Initialize (SYS_FS_HANDLE fhandle, char *file);

This function provides decoder initialize function to audio applications.

Remarks

None.

Preconditions

A FLAC audio file.

Example

```
bool flac_init_ret = bool FLAC_Initialize (flacHandle, input_buffer);
```

Parameters

Parameters	Description
fhandle	file handle to a FLAC audio file
input_buffer	input buffer pointer, size must be larger enough to hold maximum framesize of the flac file.

b) Data Types and Constants

FLAC_DEC_H Macro

File

flac_dec.h

C

```
#define FLAC_DEC_H
```

Description

This is macro FLAC_DEC_H.

Files

Files

Name	Description
flac_dec.h	This header file consists of flac library interface function declarations.

Description

This section lists the source and header files used by the FLAC Decoder Library.

flac_dec.h

This header file consists of flac library interface function declarations.

Functions

	Name	Description
	FLAC_Cleanup	A clean up function for deallocating memory resources of a FLAC decoder.

	FLAC_Decoder	An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.
	FLAC_GetBitdepth	Returns bitdepth of the FLAC audio file.
	FLAC_GetBitRate	Returns bit rate of the FLAC audio file.
	FLAC_GetBlockSize	Returns size of next packet to be decoded.
	FLAC_GetChannels	Returns number of channel of the FLAC file.
	FLAC_GetDuration	Returns track length of the FLAC audio file.
	FLAC_GetSamplingRate	Returns sample rate of the FLAC audio file.
	FLAC_Initialize	An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.
	FLAC_RegisterDecoderEventHandlerCallback	Register a decoder event handler function to FLAC decoder.
	isFLACdecoder_enabled	Return a boolean if the FLAC decoder is included or not.

Macros

	Name	Description
	FLAC_DEC_H	This is macro FLAC_DEC_H.

Description

FLAC Decoder Library Interface File

FLAC decoder interface function declarations, it is not library source code of flac, it is interface functions for easily use in Harmony audio framework.

File Name

flac_dec.h

Company

Microchip Technology Inc.

MP3 Decoder Library

This section describes the MP3 Decoder Library.

Introduction

MP3 is a lossy data compression coding format. MP3 was designed by the Moving Picture Experts Group (MPEG) as part of its MPEG-1 standard and later extended in the MPEG-2 standard. The MPLAB Harmony MP3 Decoder Library provides not only MPEG-1 standard support, but also MPEG-2 and MPEG-2.5 profile support for decoding low sampling frequency.

Description

The MP3 Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

The Compact MP3 algorithm is designed to fit in small memory footprint PIC32MX Microcontrollers. This code requires only 28 MIPS of performance (CD Quality audio), 42 KB Flash and 11 KB RAM memory for operation on the PIC32MX. This part number is for Non-modifiable binary code. Source code is also available (refer to the previous web link). Users are responsible for patent-only licensing through Technicolor.

Features

- Support for decoding MPEG-1, MPEG-2 and MPEG-2.5 encoded files
- Supports variable bit rate (VBR)
- Supports sampling frequencies from 8 kHz to 48 kHz
- Supports bit rates from 8 kbps to 320 kbps.

Library Performance

The following matrix was measured on a PIC32MX family device.

	Performance Statistics (Average)	Memory Statistics	
MP3 44.1 kHz Test Vector	MIPS	Data RAM	Flash Memory
	30	19.85 KB	47 KB

Input buffer for one MP3 Frame: 1538 byte for most cases
(for high sample frequency and bit rate audio data, 6144 byte is recommended)
Output buffer: 4608 byte for an 1152 samples, stereo 16-bit audio

Using the Library

This topic describes the basic architecture of the MP3 Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_mp3.h`

The interface to the MP3 Decoder Library is defined in the `decoder_mp3.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_mp3.h`.

Please refer to the What is MPLAB Harmony? section for how the MP3 Decoder Library interacts with the framework.

Abstraction Model

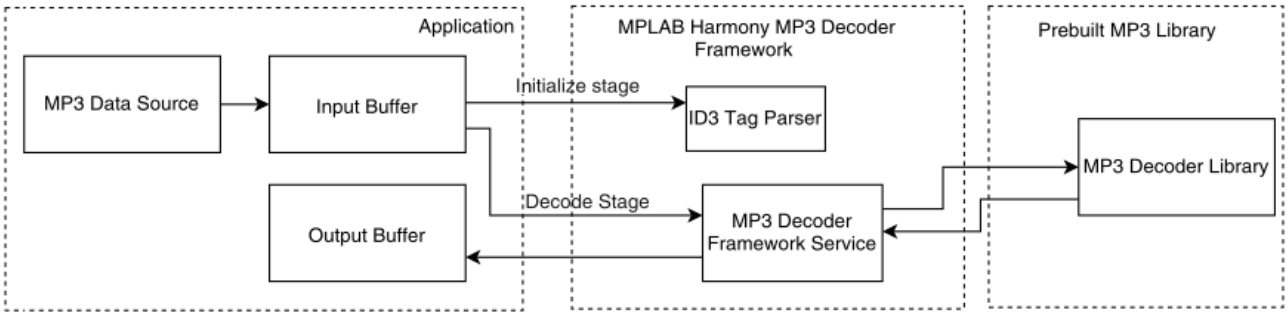
Describes the abstraction model for the MP3 Decoder Library.

Description

The following block diagram describes how the MP3 Decoder Library interacts with an application.

The MP3 Decoder Framework Service is another layer of abstraction of MP3 decoder library, it not only decodes MP3 frame, but also has a ID3 tag parser engine.

Abstraction Model



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the MP3 Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	MP3_Decode	This is function MP3_Decode.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Parse	This is function ID3_Parse.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_GetChannels	This is function MP3_GetChannels.

b) Data Types and Constants

	Name	Description
	MP3_IN_FRAME_SIZE	1538
	MP3_OUT_FRAME_SIZE	This is macro MP3_OUT_FRAME_SIZE.
	MP3_STATE_SIZE	11024
	MP3_EVENT	This is type MP3_EVENT.
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.
	MP3_STATE	This is type MP3_STATE.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.
	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame
	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame
	MP3_DEC	This is type MP3_DEC.
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.
	MP3_DEC_H	This is macro MP3_DEC_H.

Description

This section describes the Application Programming Interface (API) functions of the MP3 Decoder Library.

Refer to each section for a detailed description.

a) General Functions

MP3_Decode Function

File

mp3_dec.h

C

```
bool MP3_Decode(uint8_t * input, uint16_t inSize, uint16_t * read, uint8_t * output, uint16_t * written);
```

Description

This is function MP3_Decode.

MP3_EventHandler Function

File

mp3_dec.h

C

```
bool MP3_EventHandler(MP3_EVENT event, uint32_t data);
```

Description

This is function MP3_EventHandler.

MP3_ParseVBR Function

File

mp3_dec.h

C

```
bool MP3_ParseVBR(uint8_t* data);
```

Description

This is function MP3_ParseVBR.

isMP3decoder_enabled Function

File

mp3_dec.h

C

```
bool isMP3decoder_enabled();
```

Description

This is function isMP3decoder_enabled.

ID3_Initialize Function

File

id3.h

C

```
int32_t ID3_Initialize(uint8_t * buffer);
```

Description

void ID3_Initialize (void);

ID3_EventHandler Function

File

id3.h

C

```
bool ID3_EventHandler(ID3_EVENT event, uint32_t data);
```

Description

This is function ID3_EventHandler.

ID3_Parse Function

File

id3.h

C

```
uint32_t ID3_Parse(uint8_t * buff, uint16_t size);
```

Description

This is function ID3_Parse.

ID3_ParseFrameV22 Function

File

id3.h

C

```
void ID3_ParseFrameV22(ID3V22_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV22.

ID3_ParseFrameV23 Function

File

id3.h

C

```
void ID3_ParseFrameV23(ID3V23_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV23.

ID3_Parse_Frame Function

File

id3.h

C

```
uint32_t ID3_Parse_Frame(uint8_t * buffer, size_t left, int8_t * ret);
```

Description

This is function ID3_Parse_Frame.

MP3_GetAudioSize Function

File

mp3_dec.h

C

```
uint32_t MP3_GetAudioSize();
```

Description

This is function MP3_GetAudioSize.

MP3_UpdatePlaytime Function

File

mp3_dec.h

C

```
uint32_t MP3_UpdatePlaytime();
```

Description

This is function MP3_UpdatePlaytime.

MP3_Initialize Function

File

mp3_dec.h

C

```
bool MP3_Initialize(void * heap, uint16_t size, SYS_FS_HANDLE mp3Filehandle);
```

Description

This is function MP3_Initialize.

MP3_RegisterDecoderEventHandlerCallback Function

File

mp3_dec.h

C

```
void MP3_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);
```

Description

This is function MP3_RegisterDecoderEventHandlerCallback.

MP3_GetChannels Function

File

mp3_dec.h

C

```
uint8_t MP3_GetChannels();
```

Description

This is function MP3_GetChannels.

b) Data Types and Constants

MP3_IN_FRAME_SIZE Macro

File

mp3_dec.h

C

```
#define MP3_IN_FRAME_SIZE 6144//1538
```

Description

1538

MP3_OUT_FRAME_SIZE Macro

File

mp3_dec.h

C

```
#define MP3_OUT_FRAME_SIZE 1152 * 4
```

Description

This is macro MP3_OUT_FRAME_SIZE.

MP3_STATE_SIZE Macro

File

mp3_dec.h

C

```
#define MP3_STATE_SIZE 16876//11024
```

Description

11024

MP3_EVENT Enumeration

File

mp3_dec.h

C

```
typedef enum {
  MP3_EVENT_TAG_ARTIST,
  MP3_EVENT_TAG_ALBUM,
  MP3_EVENT_TAG_TITLE,
  MP3_EVENT_STREAM_START,
  MP3_EVENT_SAMPLERATE,
  MP3_EVENT_BITRATE,
  MP3_EVENT_TRACK_TIME
} MP3_EVENT;
```

Members

Members	Description
MP3_EVENT_TAG_ARTIST	Align TAG_ARTIST, TAG_ALBUM and TAG_TITLE with ID3 EVENT_TAGS

Description

This is type MP3_EVENT.

MP3_FRAME_HEADER Union

File

mp3_dec.h

C

```
typedef union {
  uint32_t data;
  struct {
    uint8_t sync0 : 8;
    uint8_t CRC : 1;
    uint8_t layer : 2;
    uint8_t mpeg : 2;
    uint8_t sync1 : 3;
    uint8_t padding : 1;
    uint8_t samprate : 2;
    uint8_t bitrate : 4;
    uint8_t nc : 4;
    uint8_t extend : 2;
    uint8_t stereo : 2;
  }
} MP3_FRAME_HEADER;
```

Description

This is type MP3_FRAME_HEADER.

MP3_STATE Enumeration

File

mp3_dec.h

C

```
typedef enum {
  MP3_STATE_STREAM
} MP3_STATE;
```

Description

This is type MP3_STATE.

MP3_XING_HEADER Structure

File

mp3_dec.h

C

```
typedef struct {
  int8_t tag[4];
  union {
    uint32_t flags;
    struct {
      uint32_t frameInfo : 1;
      uint32_t sizeInfo : 1;
      uint32_t tocInfo : 1;
      uint32_t qualityInfo : 1;
    }
  }
  uint8_t frames[4];
  uint8_t size[4];
  uint8_t toc[100];
  uint8_t quality;
} MP3_XING_HEADER;
```

Description

This is type MP3_XING_HEADER.

MP3_ERROR_COUNT_MAX Macro

File

mp3_dec.h

C

```
#define MP3_ERROR_COUNT_MAX 1
```

Description

This is macro MP3_ERROR_COUNT_MAX.

MP3_HEADER_CHANNELS_DUAL Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_DUAL 0b10
```

Description

This is macro MP3_HEADER_CHANNELS_DUAL.

MP3_HEADER_CHANNELS_JOINT Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_JOINT 0b01
```

Description

This is macro MP3_HEADER_CHANNELS_JOINT.

MP3_HEADER_CHANNELS_MONO Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_MONO 0b11
```

Description

This is macro MP3_HEADER_CHANNELS_MONO.

MP3_HEADER_CHANNELS_STEREO Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_STEREO 0b00
```

Description

This is macro MP3_HEADER_CHANNELS_STEREO.

MP3_HEADER_SAMPLERATE_32000 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_32000 0b10
```

Description

This is macro MP3_HEADER_SAMPLERATE_32000.

MP3_HEADER_SAMPLERATE_44100 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_44100 0b00
```

Description

This is macro MP3_HEADER_SAMPLERATE_44100.

MP3_HEADER_SAMPLERATE_48000 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_48000 0b01
```

Description

This is macro MP3_HEADER_SAMPLERATE_48000.

MP3_HEADER_SAMPLERATE_RESV Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_RESV 0b11
```

Description

This is macro MP3_HEADER_SAMPLERATE_RESV.

MP3_XING_HEADER_START_MONO Macro

File

mp3_dec.h

C

```
#define MP3_XING_HEADER_START_MONO 17
```

Description

This is macro MP3_XING_HEADER_START_MONO.

MP3_XING_HEADER_START_STEREO Macro

File

mp3_dec.h

C

```
#define MP3_XING_HEADER_START_STEREO 32
```

Description

This is macro MP3_XING_HEADER_START_STEREO.

ID3_EVENT Enumeration

File

id3.h

C

```
typedef enum {
  ID3_EVENT_TAG_ARTIST,
  ID3_EVENT_TAG_ALBUM,
  ID3_EVENT_TAG_TITLE
} ID3_EVENT;
```

Description

This is type ID3_EVENT.

ID3_STATE Enumeration

File

id3.h

C

```
typedef enum {
  ID3_STATE_INIT,
  ID3_STATE_READ_V1,
  ID3_STATE_READ_V2_HEADER,
  ID3_STATE_READ_V2_FRAME,
  ID3_STATE_FINISHED
} ID3_STATE;
```

Description

This is type ID3_STATE.

ID3V1_EXTENDED_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[4];
  int8_t title[60];
  int8_t artist[60];
  int8_t album[60];
  uint8_t speed;
  int8_t genre[30];
  int8_t startTime[6];
  int8_t endTime[6];
} ID3V1_EXTENDED_TAG;
```

Description

This is type ID3V1_EXTENDED_TAG.

ID3V1_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  int8_t title[30];
  int8_t artist[30];
  int8_t album[30];
  int8_t year[4];
  union {
    int8_t comment[30];
    struct {
      int8_t commentShort[28];
      uint8_t zero;
      uint8_t track;
    }
  }
  uint8_t genre;
} ID3V1_TAG;
```

Description

This is type ID3V1_TAG.

ID3V2_TAG_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  uint8_t version;
  uint8_t empty;
  uint8_t flag;
  uint8_t size[4];
} ID3V2_TAG_HEADER;
```

Description

This is type ID3V2_TAG_HEADER.

ID3V22_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V22_FRAME;
```

Description

This is type ID3V22_FRAME.

ID3V22_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
} ID3V22_FRAME_HEADER;
```

Description

This is type ID3V22_FRAME_HEADER.

ID3V23_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V23_FRAME;
```

Description

This is type ID3V23_FRAME.

ID3V23_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
} ID3V23_FRAME_HEADER;
```

Description

This is type ID3V23_FRAME_HEADER.

ID3_H Macro

File

id3.h

C

```
#define ID3_H
```

Description

This is macro ID3_H.

ID3_STRING_SIZE Macro

File

id3.h

C

```
#define ID3_STRING_SIZE 128
```

Description

This is macro ID3_STRING_SIZE.

ID3V22_ALBUM Macro

File

id3.h

C

```
#define ID3V22_ALBUM "TAL"
```

Description

This is macro ID3V22_ALBUM.

ID3V22_ARTIST Macro

File

id3.h

C

```
#define ID3V22_ARTIST "TP1"
```

Description

This is macro ID3V22_ARTIST.

ID3V22_TITLE Macro

File

id3.h

C

```
#define ID3V22_TITLE "TT2"
```

Description

This is macro ID3V22_TITLE.

ID3V22_ZERO Macro

File

id3.h

C

```
#define ID3V22_ZERO "\0\0\0"
```

Description

This is macro ID3V22_ZERO.

ID3V23_ALBUM Macro

File

id3.h

C

```
#define ID3V23_ALBUM "TALB"
```

Description

This is macro ID3V23_ALBUM.

ID3V23_ARTIST Macro

File

id3.h

C

```
#define ID3V23_ARTIST "TPE1"
```

Description

This is macro ID3V23_ARTIST.

ID3V23_TITLE Macro

File

id3.h

C

```
#define ID3V23_TITLE "TIT2"
```

Description

This is macro ID3V23_TITLE.

ID3V23_ZERO Macro

File

id3.h

C

```
#define ID3V23_ZERO "\0\0\0\0"
```

Description

This is macro ID3V23_ZERO.

MP3MPEG1L3_SAMPLES_PER_FRAME Macro

File

mp3_dec.h

C

```
#define MP3MPEG1L3_SAMPLES_PER_FRAME 1152 // MPEG1 LayerIII, samples per frame
```

Description

MPEG1 LayerIII, samples per frame

MP3MPEG2L3_SAMPLES_PER_FRAME Macro

File

mp3_dec.h

C

```
#define MP3MPEG2L3_SAMPLES_PER_FRAME 576  // MPEG2/2.5 LayerIII, samples per frame
```

Description

MPEG2/2.5 LayerIII, samples per frame

MP3_DEC Structure

File

mp3_dec.h

C

```
typedef struct {
  uint16_t mp3SampleRate;
  uint32_t mp3BitRate;
  uint32_t mp3Duration;
  uint32_t firstFramePos;
  bool isVBR;
  uint32_t mp3ValidBytes;
  uint8_t stereo;
} MP3_DEC;
```

Members

Members	Description
uint32_t firstFramePos;	first frame position in file
uint32_t mp3ValidBytes;	for VBR MP3, mp3ValidBytes is the number of bytes in file is given by XING header, for CBR MP3, mp3ValidBytes is MP3 audio file size - first frame position

Description

This is type MP3_DEC.

DecoderEventHandlerCB Type

File

mp3_dec.h

C

```
typedef bool (* DecoderEventHandlerCB)(uint32_t event, uint32_t data);
```

Description

This is type DecoderEventHandlerCB.

MP3_DEC_H Macro

File

mp3_dec.h

C

```
#define MP3_DEC_H
```

Description

This is macro MP3_DEC_H.

Files

Files

Name	Description
id3.h	MP3 Decoder ID3 parser header API.
mp3_dec.h	MP3 Decoder support API.

Description

This section lists the source and header files used by the MP3 Decoder Library.

id3.h

MP3 Decoder ID3 parser header API.

Enumerations

	Name	Description
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.

Functions

	Name	Description
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_Parse	This is function ID3_Parse.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.

Macros

	Name	Description
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.

Structures

	Name	Description
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.

Description

MP3 ID3 Tag Parser Header File

This file provides MP3 Decoder ID3 parser header APIs.

File Name

id3.h

Company

Microchip Technology Inc.

mp3_dec.h

MP3 Decoder support API.

Enumerations

	Name	Description
	MP3_EVENT	This is type MP3_EVENT.
	MP3_STATE	This is type MP3_STATE.

Functions

	Name	Description
	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	MP3_Decode	This is function MP3_Decode.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_GetChannels	This is function MP3_GetChannels.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.

Macros

	Name	Description
	MP3_DEC_H	This is macro MP3_DEC_H.
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.
	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_IN_FRAME_SIZE	1538
	MP3_OUT_FRAME_SIZE	This is macro MP3_OUT_FRAME_SIZE.
	MP3_STATE_SIZE	11024
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame
	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame

Structures

	Name	Description
	MP3_DEC	This is type MP3_DEC.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.

Types

	Name	Description
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.

Unions

	Name	Description
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

mp3.h

Company

Microchip Technology Inc.

Opus Decoder Library

This section describes the Opus Decoder Library.

Introduction

Introduces the Opus Decoder Library.

Description

Opus is an open, royalty-free, highly versatile audio codec. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716, which incorporated technology from Skype's SILK codec and Xiph.Org's CELT codec.

Opus Features

Supported features are:

- Bit rates from 6 kb/s to 510 kb/s
- Sampling rates from 8 kHz (narrowband) to 48 kHz (fullband)
- Frame sizes from 2.5 ms to 60 ms
- Support for both constant bit rate (CBR) and variable bit rate (VBR)
- Audio bandwidth from narrowband to fullband
- Support for speech and music
- Support for mono and stereo
- Support for up to 255 channels (mult-istream frames)
- Dynamically adjustable bit rate, audio bandwidth, and frame size
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

The full specification, RFC 6716, including the reference implementation is available from http://www.opus-codec.org. An up-to-date implementation of the Opus standard is available from the Opus Codec downloads page by visiting: https://www.opus-codec.org/downloads/

Typical Applications

- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies
- Toys
- Robots
- Any application using message playback

Resources

Feature	Option	Usage
Opus Library Flash Size	Release Build	< 143 KB
Opus Decoder RAM	Heap Size (Mono mode) Heap Size (Stereo mode)	17 KB 25 KB
Opus Decoding Performance (measured using a PIC32MX270F512L device)	16 kbps Voice 12 kbps Voice	6.8 MCPS/13.6 MIPS 4.8 MCPS/9.6 MIPS

Using the Library

This topic describes the basic architecture of the Opus Decoder Library and provides information and examples on its use.

Description

Interface Header File: `opus.h`

The interface to the Opus Decoder Library is defined in the `opus.h` header file. Any C language source (`.c`) file that uses the Opus Decoder Library should include `opus.h`.

Please refer to the What is MPLAB Harmony? section for how the Opus Decoder Library interacts with the framework.

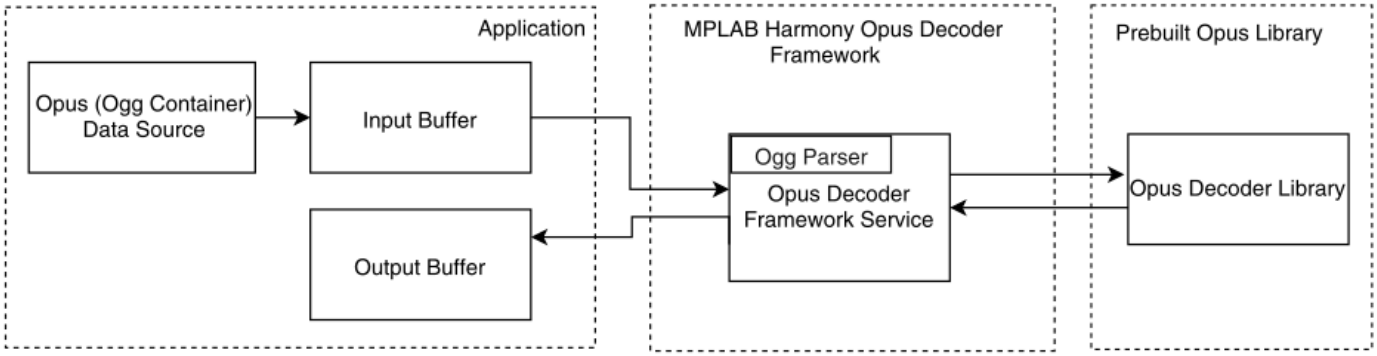
Abstraction Model

Describes the abstraction model for the Opus Decoder Library.

Description

This Opus Library is an Open Source/Free Software patent-free audio compression format designed for interactive speech and music transmission over the Internet. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Opus Decoder Library module.

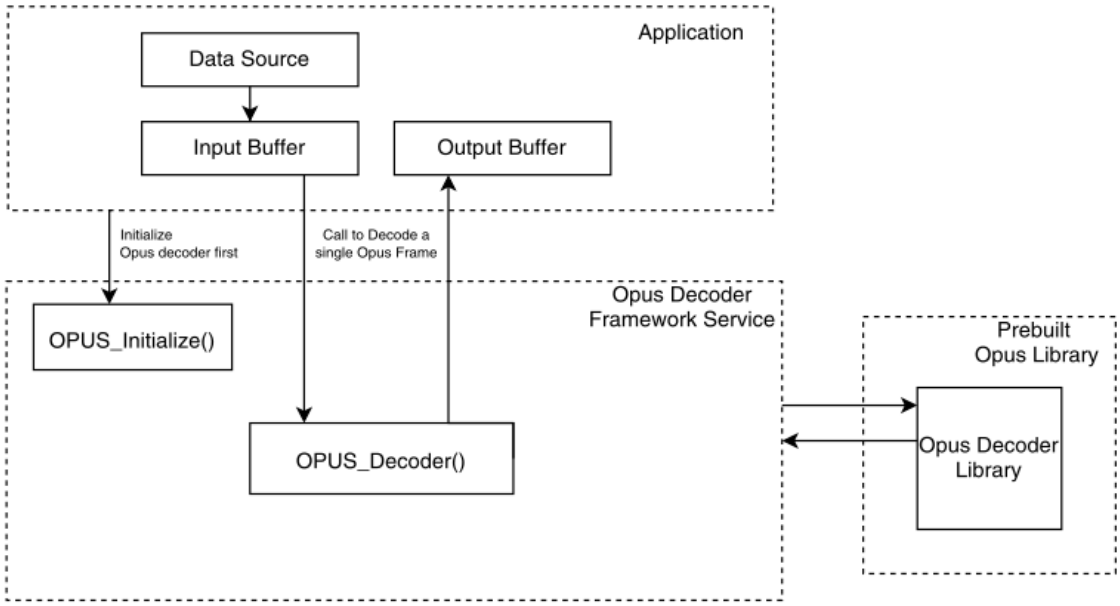
How the Library Works

This section describes how to work with the MPLAB Harmony Opus abstraction layer interface, which provides simplifier APIs for using the Opus Decoder Library.

Description

For complete documentation and how to implement Opus, visit https://www.opus-codec.org.

The following diagram describes the data flow between the Opus Decoder and an application.



Initializing the Opus Decoder

Initialize the Opus Decoder by calling the function OPUS_Initialize from the `opus_dec.h` file. This function initializes the Opus Decoder state structure. The application can start to decode Opus frame after initialization.

Decoding an Opus Frame

The OPUS_Decoder function in opus_dec.h is used to decode a single Opus frame, and write back decoded data into an output buffer.

Code Example

The following code provides an example for initializing the Opus Decoder and decoding a packet.

```
void DECODER_Initialize(decoder_type){
  switch(decoder_type){
    case OPUS:
      APP_ERROR_MSG ret = OPUS_Initialize(appDataPtr->fileHandle);
      if(ret == APP_SUCCESS)
      {
        // setup other components,
        // for example, audio CODEC, display, etc...
      }
  }
}

void App_Task()
{
  while(1){
    while(audio is not end){
      // Read one Opus packet
      appData.nBytesRead = OPUS_DiskRead(input_ptr);
     // Decode one packet
     ret = OPUS_Decoder (input_ptr,inSize,read,output,written,outBufSize);
     if(ret == success)
     {
       // continue decoding
     }else{
      // handle decoding errors;
     }
  }
    // Clean up
    OPUS_Cleanup();
    break;
  }
}
```

Library Interface

a) General Functions

	Name	Description
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

b) Data Types and Constants

	Name	Description
	OPUS_ERROR_MSG	Ogg Container Structures
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz

	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.
	DECODER_OPUS_DEC_SUPPORT_H	This is macro DECODER_OPUS_DEC_SUPPORT_H.

Description

This section describes the Application Programming Interface (API) functions of the Opus Decoder Library.
Refer to each section for a detailed description.

a) General Functions

isOPUSdecoder_enabled Function

Checks if Opus decoder is enabled by MHC configuration.

File

opus_dec.h

C

```
bool isOPUSdecoder_enabled();
```

Returns

None.

Description

The function checks if Opus decoder is enabled by MHC configuration.

Preconditions

None.

Example

```
bool opusEnabled = isOPUSdecoder_enabled ();
```

Function

bool isOPUSdecoder_enabled()

OPUS_Cleanup Function

Frees the memory allocated by the Opus Decoder.

File

opus_dec.h

C

```
void OPUS_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Opus Decoder and is called when the Opus Decoder ends.

Preconditions

None.

Example

```
OPUS_Cleanup();
```

Function

void OPUS_Cleanup()

OPUS_Decoder Function

Called once to decode one Opus packet.

File

opus_dec.h

C

```
OPUS_ERROR_MSG OPUS_Decoder(const uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output,
uint16_t * written, uint16_t outSize);
```

Returns

The status code of this function.

Description

This function is called once to decode one packet, this function will be called in an infinite while loop until the end of the packet is detected or the decode function returns failure value.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Decoder(input, inSize, read, output, written, outSize);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer, where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the packet the function writes the number of bytes consumed from the current frame decoder operation
*output	The pointer to the output sample buffer where the decided PCM samples are to be written
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer
outSize	A variable that indicates the size of output buffer, used for avoiding buffer overwritten.

Function

OPUS_ERROR_MSG OPUS_Decoder(const uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written, uint16_t outSize)

OPUS_DiskRead Function

Called once to read one Opus packet.

File

opus_dec.h

C

```
int32_t OPUS_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of this Opus packet.

Description

This function is called once to read one Opus packet.

Preconditions

None.

Example

```
int32_t bytesRead = OPUS_DiskRead(inputBuffer);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where the Opus encoded data are to be written.

Function

int32_t OPUS_DiskRead(uint8_t *inBuff)

OPUS_GetChannels Function

Returns the number of channels for this Opus audio.

File

opus_dec.h

C

```
uint8_t OPUS_GetChannels();
```

Returns

- -1 - 1 channel, mono mode
- -2 - 2 channels, stereo mode

Description

This function returns the number of channels for this Opus audio.

Preconditions

None.

Example

```
uint8_t channelNumber = OPUS_GetChannels();
```

Function

uint8_t OPUS_GetChannels()

OPUS_GetSamplingRate Function

Returns the sampling rate of Opus audio.

File

opus_dec.h

C

```
int32_t OPUS_GetSamplingRate();
```

Returns

The sampling rate of Opus audio.

Description

This function returns the sampling rate of this Opus file, return value is always 48000, Opus playback sample rate should be 48000 as described in the OggOpus spec.

Preconditions

None.

Example

```
int32_t sampleRate = OPUS_GetSamplingRate();
```

Function

int32_t OPUS_GetSamplingRate()

OPUS_Initialize Function

Initializes the Opus decoder and creates an Opus decoder state structure.

File

opus_dec.h

C

```
OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler);
```

Returns

The status code of this function.

Description

This function initializes the Opus decoder and creates an Opus decoder state structure.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Initialize(fileHandler);
```

Parameters

Parameters	Description
opus_file_handler	The file handler of current Opus file which will be decoded.

Function

OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler)

b) Data Types and Constants

OPUS_ERROR_MSG Enumeration

File

opus_dec.h

C

```
typedef enum {
  OPUS_SUCCESS = 1,
  OPUS_READ_ERROR,
  OPUS_STREAM_ERROR,
  OPUS_BUFF_ERROR,
  OPUS_STREAM_END,
  OPUS_PLAYBACK_ERROR,
  OPUS_OUT_OF_MEM_ERROR,
  OPUS_DISK_ERROR,
  OPUS_GENERAL_ERROR
} OPUS_ERROR_MSG;
```

Description

Ogg Container Structures

opusDecDcpt Structure

File

opus_dec.h

C

```
typedef struct {
  int processedPktNo;
```

```
  int currPktNo;
  int nInBytes;
} opusDecDcpt;
```

Members

Members	Description
int processedPktNo;	counter of processed packets
int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type opusDecDcpt.

sOggPageHdr Structure

File

opus_dec.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
  uint8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
uint8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

sOpusHeader Structure

File

opus_dec.h

C

```
typedef struct {
  char signature[8];
  uint8_t version;
  uint8_t channels;
  uint16_t preskip;
  uint32_t input_sample_rate;
  uint16_t gain;
  uint8_t channel_mapping;
  int8_t nb_streams;
  int8_t nb_coupled;
  unsigned char stream_map[255];
```

```
} sOpusHeader;
```

Members

Members	Description
char signature[8];	Magic signature: "OpusHead
uint8_t version;	Version number: 0x01 for this spec
uint8_t channels;	Number of channels: 1..255
uint16_t preskip;	This is the number of samples (at 48kHz) to discard from the decoder output when starting playback
uint32_t input_sample_rate;	Original input sample rate in Hz, this is not the sample rate to use for playback of the encoded data
uint16_t gain;	output gain in dB
uint8_t channel_mapping;	This byte indicates the order and semantic meaning of various channels encoded in each Opus packet The rest is only used if channel_mapping != 0
int8_t nb_streams;	This field indicates the total number of streams so the decoder can correctly parse the packed Opus packets inside the Ogg packet
int8_t nb_coupled;	Describes the number of streams whose decoders should be configured to produce two channels

Description

This is type sOpusHeader.

sOpusPktDcpt Structure

File

opus_dec.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sOpusPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sOpusStreamDcpt Structure

File

opus_dec.h

C

```
typedef struct {
  sOggPageHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sOpusStreamDcpt;
```

Members

Members	Description
sOggPageHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

OPUS_INPUT_BUFFER_SIZE Macro

File

opus_dec.h

C

```
#define OPUS_INPUT_BUFFER_SIZE (1024*2)
```

Description

MACROS

OPUS_MAX_FRAME_SIZE Macro

File

opus_dec.h

C

```
#define OPUS_MAX_FRAME_SIZE (960*6) // 120ms @ 48Khz
```

Description

120ms @ 48Khz

OPUS_OUTPUT_BUFFER_SIZE Macro

File

opus_dec.h

C

```
#define OPUS_OUTPUT_BUFFER_SIZE (1024*7)
```

Description

This is macro OPUS_OUTPUT_BUFFER_SIZE.

DECODER_OPUS_DEC_SUPPORT_H Macro

File

opus_dec.h

C

```
#define DECODER_OPUS_DEC_SUPPORT_H
```

Description

This is macro DECODER_OPUS_DEC_SUPPORT_H.

Files

Files

Name	Description
opus_dec.h	Contains the Opus decoder specific defintions and function prototypes.

Description

This section lists the source and header files used by the Opus Decoder Library.

opus_dec.h

Contains the Opus decoder specific defintions and function prototypes.

Enumerations

	Name	Description
	OPUS_ERROR_MSG	Ogg Container Structures

Functions

	Name	Description
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

Macros

	Name	Description
	DECODER_OPUS_DEC_SUPPORT_H	This is macro DECODER_OPUS_DEC_SUPPORT_H.
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz
	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.

Structures

	Name	Description
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time

Description

Ogg-Opus Decoder Interface File

This file contains the opus decoder specific defintions and function prototypes.

File Name

opus_dec.h

Company

Microchip Technology Inc.

Speex Decoder Library

This section describes the Speex Decoder Library.

Introduction

Speex is an Open Source/Free Software patent-free audio compression format designed for speech. It is designed by the Xiph.Org Foundation, and has been obsoleted by Opus.

Description

Speex is a Code Excited Linear Prediction (CELP) based open source patent-free audio compression format designed for speech.

Speex Features

- Fixed-point implementation

- Narrowband (8 kHz) and wideband (16 kHz) bit-streams
- Wide range of bit-rates available (from 2.15 kbps to 44 kbps)
- Perceptual enhancement (attempts to reduce the perception of the noise/distortion produced by the encoding/decoding process. In most cases, perceptual enhancement brings the sound further from the original objectively (e.g. considering only SNR), but in the end it still sounds better (subjective improvement)

Visit www.speex.org for further details and complete documentation

Typical Applications

- Answering machines; Voice recorders
- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies; Toys and robots
- Any application using message playback

Resources

Feature	Option	Usage
Speex Library Flash Size	Release Build	< 60 KB
	Debug Build	< 92 KB
Speex Decoder RAM	Dynamic	4 KB
	Stack	2 KB
Speex Decoding Performance	Narrow Mode	280 KB/sec

Using the Library

This topic describes the basic architecture of the Speex Decoder Library and provides information and examples on its use.

Description

Interface Header File: `speex.h`

The interface to the Speex Decoder Library is defined in the `speex.h` header file. Any C language source (`.c`) file that uses the Speex Decoder Library should include `speex.h`.

Please refer to the What is MPLAB Harmony? section for how the Speex Decoder Library interacts with the framework.

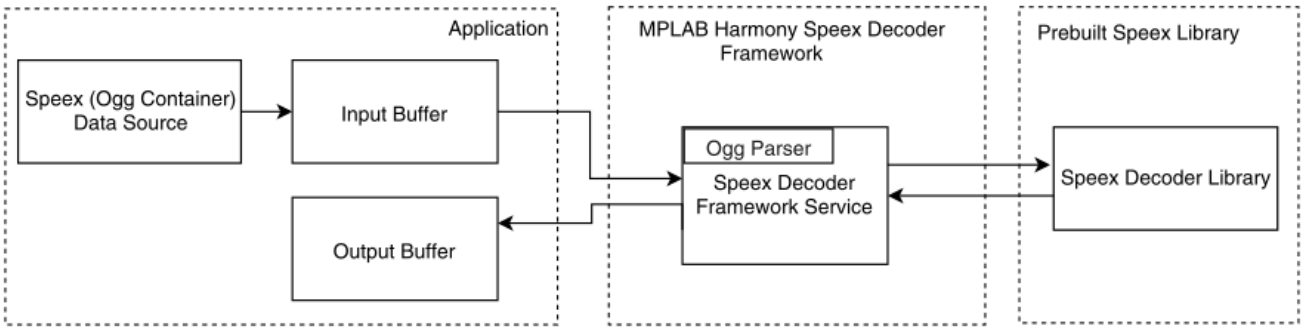
Abstraction Model

Describes the abstraction model for the Speex Decoder Library.

Description

This Speex Library is an Open Source/Free Software patent-free audio compression format designed for speech. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Speex Decoder Library module.

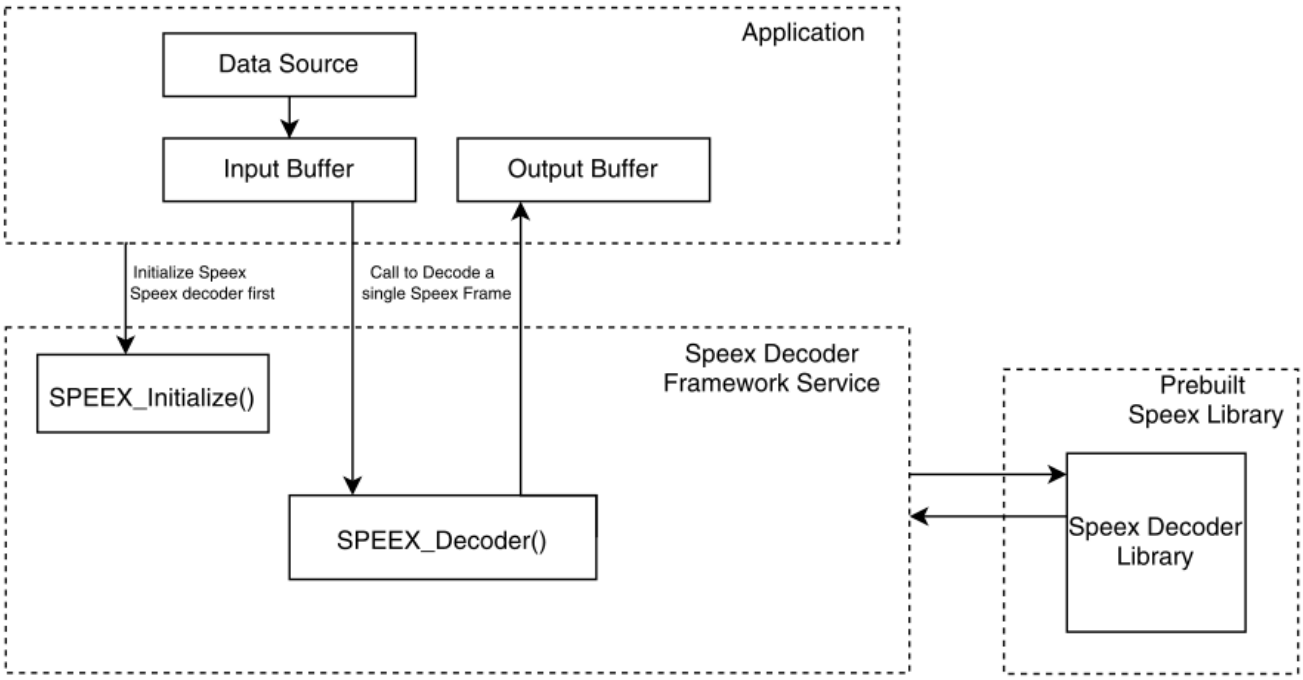
How the Library Works

This section describes how to work with the MPLAB Harmony Speex abstraction layer interface, which provides simplifier APIs for using the Speex Decoder Library.

Description

For complete documentation and how to implement Speex, visit www.speex.org.

The following diagram describes the data flow between the Speex Decoder and an application.



Initializing the Speex Decoder

Initialize the Speex Decoder by calling the function SPEEX_Initialize from the `speex_dec.h` file. This function initializes the Speex Decoder state structure. The application can start to decode a Speex frame after initialization.

Decoding a Speex Frame

The SPEEX_Decoder function in `speex_dec.h` decodes a single Speex frame, and writes back decoded data into an output buffer.

Code Example

The following code provides an example for initializing the Speex Decoder Library and reading and decoding one Speex packet.

```
void Decoder_Initialize(decoder_type){
    switch (decoder_type){
        case SPEEX:
            ret = SPEEX_Initialize();
            if(ret == success)
            {
                // start decoding
            }
    }
}

void App_Task()
{
    while(1){
        while(audio is not end){
            // Read one Speex packet
            appData.nBytesRead = SPEEX_DiskRead(input_ptr);

            // Decode one packet
            ret = SPEEX_Decoder (input_ptr,inSize,read,output,written);
            if(ret == success)
```

```
            {
                // continue decoding
            }else{
                // handle decoding errors;
            }
        }
        // Clean up
        SPEEX_Cleanup();
    }

}
```

Library Interface

a) General Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.
	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.
	SPEEX_GetChannels	Returns channel number of Ogg-Speex audio.

b) Data Types and Constants

	Name	Description
	eSpxFlags	Speex flags
	pProgressFnc	progress display function
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Description

This section describes the Application Programming Interface (API) functions of the Speex Decoder Library.

Refer to each section for a detailed description.

a) General Functions

isSPEEXdecoder_enabled Function

Indicates whether the Speex Decoder is enabled.

File

speex_dec.h

C

```
bool isSPEEXdecoder_enabled();
```

Returns

This function returns a Boolean value.

- true - Indicates that the Speex Decoder is enabled
- false - Indicates that the Speex Decoder is disabled

Description

This function indicates whether the Speex Decoder is enabled.

Preconditions

None.

Example

```
appData.SPEEX_decoder_enabled   = isSPEEXdecoder_enabled();
```

Function

bool isSPEEXdecoder_enabled()

SPEEX_Cleanup Function

Frees the memory allocated by the Speex Decoder.

File

speex_dec.h

C

```
void SPEEX_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Speex Decoder and is called when the Speex Decoder ends.

Preconditions

None.

Example

```
SPEEX_Cleanup();
```

Function

void SPEEX_Cleanup()

SPEEX_Decoder Function

Called once to decode one packet.

File

speex_dec.h

C

```
SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t
```

```
* written);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns failure value.

Preconditions

None.

Example

```
res = SPEEX_Decoder (input,inSize,read,output,written);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer , where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*output	The pointer to the output sample buffer where the decided PCM samples are to be written,
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written);

SPEEX_DiskRead Function

Reads one packet of Ogg-Speex audio.

File

speex_dec.h

C

```
int32_t SPEEX_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of reading bytes.

Description

This function is called once to read one packet.

Preconditions

None.

Example

```
appData.nBytesRead = SPEEX_DiskRead(ptr);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where one packet data is to to be written

Function

int32_t SPEEX_DiskRead(uint8_t *inBuff)

SPEEX_GetBitrate Function

Returns the bit rate of Ogg-Speex audio.

File

speex_dec.h

C

```
int32_t SPEEX_GetBitrate();
```

Returns

This function returns the bit rate.

Description

This function returns the bit rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
decoderBitrate = SPEEX_GetBitrate()/1000;
```

Function

int32_t SPEEX_GetBitrate()

SPEEX_GetSamplingRate Function

Returns the sampling rate of Ogg-Speex audio.

File

speex_dec.h

C

```
int32_t SPEEX_GetSamplingRate();
```

Returns

This function returns the sampling rate.

Description

This function returns the sampling rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
sampling_frequency = SPEEX_GetSamplingRate();
```

Function

int32_t SPEEX_GetSamplingRate()

SPEEX_Initialize Function

Reads the Speex header page and initializes the Speex Decoder state.

File

speex_dec.h

C

```
SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

The function internally calls the Speex Codec Decoder initialization function. The function is called only once before initiating the decoding process.

Preconditions

Call SPEEX_Cleanup function before this function for safety.

Example

```
APP_ERROR_MSG res = SPEEX_Initialize(spx_file_handle);
```

Function

SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);

SPEEX_GetChannels Function

Returns channel number of Ogg-Speex audio.

File

speex_dec.h

C

```
uint8_t SPEEX_GetChannels();
```

Returns

This function returns channel number

Description

This function returns the channel number of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
channelNumber = SPEEX_GetChannels();
```

Function

uint8_t SPEEX_GetChannels()

b) Data Types and Constants

eSpxFlags Enumeration

File

speex_dec.h

C

```
typedef enum {
  SPX_FLAG_WB = 0x1,
  SPX_FLAG_VBR = 0x2,
  SPX_FLAG_BRATE_OVR = 0x4,
  SPX_FLAG_PRCPT_ENH = 0x8
```

```
} eSpxFlags;
```

Members

Members	Description
SPX_FLAG_WB = 0x1	wide/narrow band
SPX_FLAG_VBR = 0x2	VBR
SPX_FLAG_BRATE_OVR = 0x4	bit rate setting overrides the encoder quality
SPX_FLAG_PRCPT_ENH = 0x8	decoder perceptual enhancement on/off

Description

Speex flags

Remarks

at most 8 flags are supported right now!

pProgressFnc Type

File

speex_dec.h

C

```
typedef void (* pProgressFnc)(int nBytes);
```

Description

progress display function

progressDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  int progressStep;
  int progressCnt;
  pProgressFnc progressFnc;
} progressDcpt;
```

Members

Members	Description
int progressStep;	no of bytes to process before calling the progress callback
int progressCnt;	current counter
pProgressFnc progressFnc;	progress callback

Description

encoder/decoder progress activity descriptor

sOggPageSegHdr Structure

File

speex_dec.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
```

```
  int8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageSegHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
int8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

SPEEX_ERROR_MSG Enumeration

File

speex_dec.h

C

```
typedef enum {
  SPEEX_SUCCESS = 1,
  SPEEX_READ_ERROR,
  SPEEX_STREAM_ERROR,
  SPEEX_BUFF_ERROR,
  SPEEX_STREAM_END,
  SPEEX_PLAYBACK_ERROR,
  SPEEX_OUT_OF_MEM_ERROR,
  SPEEX_DISK_ERROR,
  SPEEX_GENERAL_ERROR
} SPEEX_ERROR_MSG;
```

Description

This is type SPEEX_ERROR_MSG.

spxCdcDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  SpeexBits spxBits;
  void* spxState;
  int inBuffSize;
  int outBuffSize;
  int nOutBytes;
} spxCdcDcpt;
```

Members

Members	Description
SpeexBits spxBits;	codec control structure
void* spxState;	codec state
int inBuffSize;	input buffer size
int nOutBytes;	bytes available in the output buffer

Description

Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part

spxDecDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  spxCdcDcpt cdc;
  int framesPerPacket;
  int frameSize;
  int processedPktNo;
  int currPktNo;
  int nInBytes;
  int outFrameSize;
} spxDecDcpt;
```

Members

Members	Description
spxCdcDcpt cdc;	common part
int framesPerPacket;	frames per Ogg packet
int frameSize;	size of the frames
int processedPktNo;	counter of processed packets
int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type spxDecDcpt.

sSpeexHeader Structure

File

speex_dec.h

C

```
typedef struct {
  char speexString[SPEEX_STRING_LENGTH];
  char speexVer[SPEEX_VERSION_LENGTH];
  int32_t speexVerId;
  int32_t headerSize;
  int32_t sampleRate;
  int32_t wBand;
  int32_t modeBitsStreamVer;
  int32_t nChannels;
  int32_t bitRate;
  int32_t frameSamples;
  int32_t vbr;
  int32_t framesPerPacket;
  int32_t extraHeaders;
  int32_t reserved1;
  int32_t reserved2;
} sSpeexHeader;
```

Members

Members	Description
char speexString[SPEEX_STRING_LENGTH];	identify the Speex bit-stream: OGG_ID_SPEEX
char speexVer[SPEEX_VERSION_LENGTH];	Speex version that encoded the file
int32_t speexVerId;	Speex version (for checking compatibility)
int32_t headerSize;	sizeof(SpeexHeader)

int32_t sampleRate;	Sampling rate used
int32_t wBand;	0 for narrowband, 1 for wideband
int32_t modeBitsStreamVer;	Version ID of the bit-stream
int32_t nChannels;	Number of channels encoded
int32_t bitRate;	Bit-rate used
int32_t frameSamples;	Size of frames, samples
int32_t vbr;	1 for a VBR encoding, 0 otherwise
int32_t framesPerPacket;	Number of frames stored per Ogg packet
int32_t extraHeaders;	Number of additional headers after the comments
int32_t reserved1;	Reserved for future use, 0
int32_t reserved2;	Reserved for future use, 0

Description

the Speex header, the first page in the Speex stream

sSpxPktDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sSpxPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sSpxRunDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  int streamNo;
  int streamVer;
  unsigned short frameSamples;
  unsigned short bitRate;
  unsigned char framesPerPacket;
  unsigned char packetsPerPage;
  unsigned char complexity;
  unsigned char qualFactor;
  char spxFlags;
  char reserved[3];
} sSpxRunDcpt;
```

Members

Members	Description
int streamNo;	stream number inside the container
int streamVer;	stream version
unsigned short frameSamples;	Size of frames, in samples
unsigned short bitRate;	encoder bit rate
unsigned char framesPerPacket;	Number of frames stored per Ogg packet, <10

unsigned char packetsPerPage;	number of packets in an Ogg page <= 255
unsigned char complexity;	encoder complexity 1-10
unsigned char qualFactor;	encoder quality factor 1-10
char spxFlags;	eSpxFlags: run time flags, wideband, VBR
char reserved[3];	future expansion/padding

Description

run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()

sSpxStreamDcpt Structure

File

speex_dec.h

C

```
typedef struct {
  sSpxRunDcpt runDcpt;
  sOggPageSegHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sSpxStreamDcpt;
```

Members

Members	Description
sSpxRunDcpt runDcpt;	global info
sOggPageSegHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

OGG_ID_SPEEX Macro

File

speex_dec.h

C

```
#define OGG_ID_SPEEX "Speex   "    // The Speex packet ID
```

Description

The Speex packet ID

SPEEX_INPUT_BUFFER_SIZE Macro

File

speex_dec.h

C

```
#define SPEEX_INPUT_BUFFER_SIZE 1024
```

Description

This is macro SPEEX_INPUT_BUFFER_SIZE.

SPEEX_OUTPUT_BUFFER_SIZE Macro

File

speex_dec.h

C

```
#define SPEEX_OUTPUT_BUFFER_SIZE 1024*7
```

Description

This is macro SPEEX_OUTPUT_BUFFER_SIZE.

SPEEX_STRING_LENGTH Macro

File

speex_dec.h

C

```
#define SPEEX_STRING_LENGTH 8            // The size of the Speex string
```

Description

The size of the Speex string

SPEEX_VENDOR_STR Macro

File

speex_dec.h

C

```
#define SPEEX_VENDOR_STR "Encoded by Microchip Audio Library ver 1.0"    // comment in Ogg header
```

Description

comment in Ogg header

SPEEX_VERSION Macro

File

speex_dec.h

C

```
#define SPEEX_VERSION "speex-1.2beta3"    // The Speex version string
```

Description

The Speex version string

SPEEX_VERSION_ID Macro

File

speex_dec.h

C

```
#define SPEEX_VERSION_ID 1                    // Version identifier
```

Description

Version identifier

SPEEX_VERSION_LENGTH Macro

File

speex_dec.h

C

```
#define SPEEX_VERSION_LENGTH 20                // The size of the Speex version string
```

Description

The size of the Speex version string

SPX_CODEC_BUFF_DIV Macro

File

speex_dec.h

C

```
#define SPX_CODEC_BUFF_DIV 5    // 20% is sufficient
```

Description

20% is sufficient

SPX_CODEC_BUFF_MUL Macro

File

speex_dec.h

C

```
#define SPX_CODEC_BUFF_MUL 6    // Values to adjust the average decoder buffer size
```

Description

Values to adjust the average decoder buffer size

Files

Files

Name	Description
speex_dec.h	This file is an abstraction layer of the Speex Library.

Description

This section lists the source and header files used by the Speex Decoder Library.

speex_dec.h

This file is an abstraction layer of the Speex Library.

Enumerations

	Name	Description
	eSpxFlags	Speex flags
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.

Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.

	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetChannels	Returns channel number of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.

Macros

	Name	Description
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_DEC_SUPPORT_H	This is macro SPEEX_DEC_SUPPORT_H.
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Structures

	Name	Description
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time

Types

	Name	Description
	pProgressFnc	progress display function

Description

Speex Decoder Library Interface Definition

This file contains the Speex Decoder-specific definitions and function prototypes.

File Name

speex_dec.h

Company

Microchip Technology Inc.

WMA Decoder Library

This section describes the WMA Decoder Library.

Introduction

Windows Media Audio (WMA) is the name of a series of Audio Codecs and their corresponding audio coding formats developed by Microsoft. It is a proprietary technology that forms part of the Windows Media framework.

Description

The WMA Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. The library is only available in binary format, and is only available to Microsoft Windows Media Component Licensees. For licensing

information, please visit: http://windows.microsoft.com/en-us/windows/windowsmedia-components-licensing. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

Windows Media Audio (WMA) developed by Microsoft, is a format enabling the storage of digital audio using the Lossy compression algorithm.

The Microchip WMA Decoder can decode audio signals sampled at up to 48 kHz with up to two discrete channels. The WMA Decoder also supports VBR and CBR encoded audio stream. In most circumstances, `.wma` files are contained in Advance Systems Format (ASF), which is supported by the WMA Decoder. The WMA Decoder library is optimized (C/ASM) and is available for all PIC32MX devices.

Features

- Supports both variable bit rate (VBR) and constant bit rate (CBR)
- Supports WMA version v9.2, v9.1, v9, v8, v7, v4.1, v4.0
- Supports sampling frequency range from 8 kHz to 48 kHz
- Supports bit rate range from 128 bps to 384 kbps

Library Performance

	Performance Statistics (Average)	Memory Statistics	
WMA 44.1 kHz, 192 kbps Test Vector	MIPS	Data RAM	Flash Memory
	35	32.23KB	102.72KB

Input buffer for one WMA frame: 20 Kbytes

Output buffer: 12 Kbytes for stereo 16-bit audio data

Using the Library

This topic describes the basic architecture of the WMA Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_wma.h`

The interface to the WMA Decoder Library is defined in the `decoder_wma.h` header file. Any C language source (`.c`) file that uses the WMA Decoder Library should include `decoder_wma.h`.

Please refer to the What is MPLAB Harmony? section for how the WMA Decoder Library interacts with the framework.

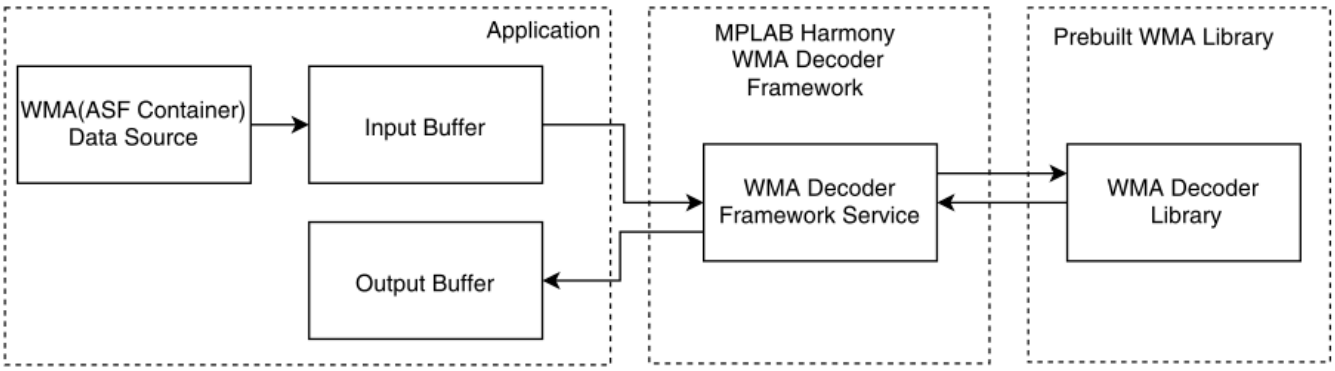
Abstraction Model

Describes the abstraction model for the WMA Decoder Library.

Description

The following figure describes the abstraction model for the WMA Decoder Library.

Abstraction Model



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the WMA Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	WMA_Decoder	This is function WMA_Decoder.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_GetChannels	This is function WMA_GetChannels.

b) Data Types and Constants

	Name	Description
	SYS_DEBUG_BUFFER_DMA_READY	This should be defined in system_config.h. It is added here as a build safe-guard.
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.
	SPEEX_DEC_SUPPORT_H	This is macro SPEEX_DEC_SUPPORT_H.

Description

This section describes the Application Programming Interface (API) functions of the WMA Decoder Library.

Refer to each section for a detailed description.

a) General Functions

WMA_Decoder Function

File

wma_dec.h

C

```
int16_t WMA_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
written);
```

Description

This is function WMA_Decoder.

WMA_SamplingFrequency_Get Function

File

wma_dec.h

C

```
int32_t WMA_SamplingFrequency_Get();
```

Description

This is function WMA_SamplingFrequency_Get.

WMA_FreeMemory Function

File

wma_dec.h

C

```
void WMA_FreeMemory();
```

Description

This is function WMA_FreeMemory.

isWMAdecoder_enabled Function

File

wma_dec.h

C

```
bool isWMAdecoder_enabled();
```

Description

This is function isWMAdecoder_enabled.

WMA_BitRate_Get Function

File

wma_dec.h

C

```
int32_t WMA_BitRate_Get();
```

Description

This is function WMA_BitRate_Get.

WMA_GetHeaderPacketOffset Function

File

wma_dec.h

C

```
int32_t WMA_GetHeaderPacketOffset();
```

Description

This is function WMA_GetHeaderPacketOffset.

WMA_Initialize Function

File

wma_dec.h

C

```
void WMA_Initialize(SYS_FS_HANDLE wmaFilehandle, uint32_t inputBufferSize);
```

Description

This is function WMA_Initialize.

WMA_RegisterAppCallback Function

File

wma_dec.h

C

```
void WMA_RegisterAppCallback(SetReadBytesReadFlagInAppData fptr0, GetReadBytesInAppData fptr1);
```

Description

This is function WMA_RegisterAppCallback.

WMA_GetChannels Function

File

wma_dec.h

C

```
uint8_t WMA_GetChannels();
```

Description

This is function WMA_GetChannels.

b) Data Types and Constants

SYS_DEBUG_BUFFER_DMA_READY Macro

File

sys_debug.h

C

```
#define SYS_DEBUG_BUFFER_DMA_READY
```

Description

This should be defined in system_config.h. It is added here as a build safe-guard.

WMA_DECODER_STATES Enumeration

File

wma_dec.h

C

```
typedef enum {
  WMA_GET_FRAME_SIZE,
  WMA_DECODE_FRAME
} WMA_DECODER_STATES;
```

Description

This is type WMA_DECODER_STATES.

WMA_ERROR_COUNT_MAX Macro

File

wma_dec.h

C

```
#define WMA_ERROR_COUNT_MAX 1
```

Description

This is macro WMA_ERROR_COUNT_MAX.

WMA_H Macro

File

wma_dec.h

C

```
#define WMA_H
```

Description

This is macro WMA_H.

GetReadBytesInAppData Type

File

wma_dec.h

C

```
typedef int32_t (* GetReadBytesInAppData)();
```

Description

This is type GetReadBytesInAppData.

SetReadBytesReadFlagInAppData Type

File

wma_dec.h

C

```
typedef void (* SetReadBytesReadFlagInAppData)(int32_t val, bool b);
```

Description

This is type SetReadBytesReadFlagInAppData.

SPEEX_DEC_SUPPORT_H Macro

File

speex_dec.h

C

```
#define SPEEX_DEC_SUPPORT_H
```

Description

This is macro SPEEX_DEC_SUPPORT_H.

Files

Files

Name	Description
wma_dec.h	WMA Decoder support API.

Description

This section lists the source and header files used by the MP3 Decoder Library.

wma_dec.h

WMA Decoder support API.

Enumerations

	Name	Description
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.

Functions

	Name	Description
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_Decoder	This is function WMA_Decoder.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	WMA_GetChannels	This is function WMA_GetChannels.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.

Macros

	Name	Description
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.

Types

	Name	Description
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

wma.h

Company

Microchip Technology Inc.

Driver Libraries Help

This section provides descriptions of the Driver libraries that are available in MPLAB Harmony.

Driver Library Overview

This topic provides help for the MPLAB Harmony driver libraries. It includes a general driver usage overview, as well as sections providing a programmer's reference for each driver that describes its interface and explains how to use it.

Introduction

Introduces MPLAB Harmony device drivers and explains common usage concepts.

Description

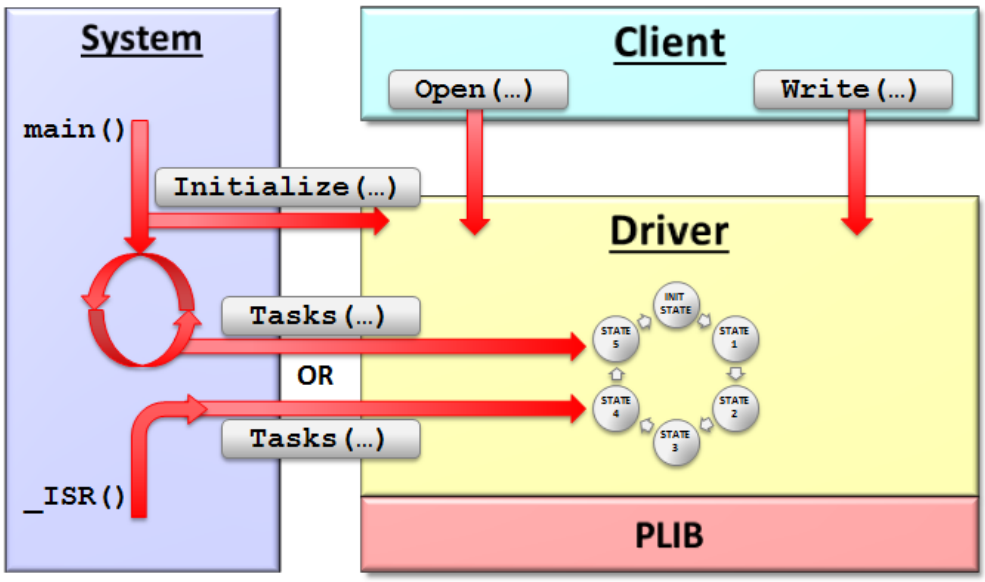
MPLAB Harmony device drivers (usually referred to as "drivers") provide simple, highly abstracted C-language interfaces to peripherals and other resources. A driver's interface allows applications and other client modules to easily interact with the peripheral it controls using consistent usage models. Some functions are similar on all drivers, while other functions are unique to a particular type of driver or peripheral. However, driver interface functions are generally independent of the details of how a given peripheral is implemented on any specific hardware or of how many instances of that peripheral exist in a given system.

Drivers normally utilize MPLAB Harmony Peripheral Libraries (PLIBs) to access and control peripheral hardware that is built into the processor (and is directly addressable by it). However, drivers can also support external peripheral hardware by calling another driver that directly controls a built-in peripheral to which the external peripheral is connected. For example, an SD Card driver may use a SPI driver to access its external SD Card Flash device. A driver may even be completely abstracted away from any hardware (utilizing no peripheral hardware at all), simply controlling some software resource (such as a buffer queue) or providing some service (such as data formatting or encryption). Using this method, driver and other modules may be "stacked" into layers of software, with each responsible for the details of managing its own resources while hiding those details from client modules that use them.

Regardless of the type of peripheral or resource that a MPLAB Harmony driver manages, a driver has the following fundamental responsibilities:

- Provide a common system-level interface to the resource
- Provide a highly abstracted file system style client interface to the resource
- Manage the state of the peripheral or resource
- Manage access to the resource

A driver's system interface can be thought of as being a horizontal interface and its client interface can be thought of as being a vertical interface, as shown in the following block diagram.



The horizontal or "system" interface provides functions to initialize the driver and keep it running. To keep a driver running, a system loop or ISR function (but never both in the same system) calls its state machine "tasks" function repeatedly, as necessary. Therefore, a driver's system interface is normally only called by code that is generated by the MPLAB Harmony Configurator (MHC) when you select and configure the driver. Its purpose is to ensure that the driver works independently (conceptually in the background), providing the capabilities it implements. By contrast, the application (or any other "client" of the driver) normally only interacts with the driver's vertical "client" interface (often thought of as the driver's API). The client interface provides functions to open the driver for use and interact with it, reading or writing data or performing device-type specific operations. The client interface is what allows the application to access the peripheral in a safe and easy way without worrying about the details of the driver or what other clients it may be serving.

The following sections describe in general terms how to use these two interfaces and give specific examples to help illustrate the concepts. The

subsequent help sections for each individual driver describe their specific interfaces in detail; listing all supported functions, parameters, and return values as well as their data types and expected behavior. You may also refer to the MPLAB Harmony Driver Development guide for additional information on MPLAB Harmony drivers and for information on how to develop your own drivers, if needed.

Using a Driver's System Interface

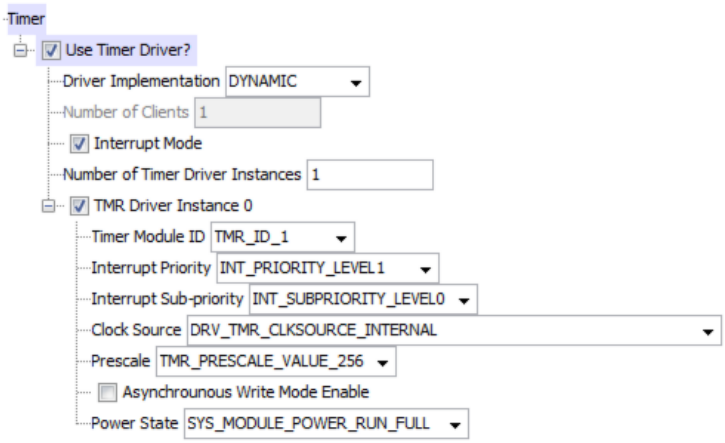
Introduces the System Interface of a MPLAB Harmony device driver and explains its usage.

Description

An MPLAB Harmony driver's system interface provides functions to initialize, deinitialize, and reinitialize an instance of a driver, as well as functions to maintain its state machine (and/or implement its Interrupt Service Routine) and check its current "running" status. Normally, as an MPLAB Harmony application developer or a developer of a "client" module that uses the driver, you will not call the system interface functions directly. The MHC generates calls to the system interface functions of any driver that is used in a project when it generates the system configuration files. Exactly which functions are called and exactly how they're called depends on the configuration options selected in the project's active configuration.

For example, when the box next to "Use Timer Driver?" is selected in the MHC Options tree (within *MPLAB Harmony & Application Configuration* > *Harmony Framework Configuration* > *Drivers* > *Timer*), as shown in the following figure, the MHC will generate all necessary definitions and function calls for the Timer Driver's system interface.

Example Timer Driver MHC Options



These configuration selections, which are set by default once "Use Timer Driver" is selected, will cause the MHC to generate the following definitions in the `system_config.h` header file for the main project's current configuration when **Generate Code** is clicked.

Example Driver Options in system_config.h

```
/*** Timer Driver Configuration ***/
#define DRV_TMR_INTERRUPT_MODE            true
#define DRV_TMR_INSTANCES_NUMBER          1
#define DRV_TMR_CLIENTS_NUMBER            1

/*** Timer Driver 0 Configuration ***/
#define DRV_TMR_PERIPHERAL_ID_IDX0         TMR_ID_1
#define DRV_TMR_INTERRUPT_SOURCE_IDX0      INT_SOURCE_TIMER_1
#define DRV_TMR_CLOCK_SOURCE_IDX0          DRV_TMR_CLKSOURCE_INTERNAL
#define DRV_TMR_PRESCALE_IDX0              TMR_PRESCALE_VALUE_256
#define DRV_TMR_OPERATION_MODE_IDX0        DRV_TMR_OPERATION_MODE_16_BIT
#define DRV_TMR_ASYNC_WRITE_ENABLE_IDX0    false
#define DRV_TMR_POWER_STATE_IDX0           SYS_MODULE_POWER_RUN_FULL
```

It is important to notice that the Driver Implementation selection in the MHC graphical interface does not correlate to a `#define` statement in the `system_config.h` file. Instead, it determines which implementation of the driver this configuration will use. Drivers may have more than one implementation. For example, most drivers have both static and dynamic implementations. A static implementation is usually the smaller of the two, but it is only capable of controlling one instance of a peripheral. An equivalent dynamic implementation will be larger, but it is capable of managing multiple instances of the same type of peripheral using a single instance of the source code (and thus, one instance of the object code). Some drivers may have additional implementations, each one optimized for a different usage. The Driver Implementation pull-down control in the MHC graphical interface allows you to select which implementation the current configuration will use. Normally, you can use only a single implementation of a driver in a given configuration. If you change driver implementations, it changes which implementation is used for all all instances of a peripheral.

The number of instances option, for example, Number of Timer Driver Instances, which correlates to the DRV_TMR_INSTANCES_NUMBER definition, determines how many instances of a static driver implementation will be generated or how many instances of a peripheral a dynamic driver implementation will manage. Drivers may also be designed to allow multiple different clients (applications or other modules) to share the same instance of a peripheral or resource. Therefore, a driver will have an option to determine a maximum number of simultaneous clients that it can support. For example, Number of Clients (DRV_TMR_CLIENTS_NUMBER) in the Timer Driver, which is fixed at one (1) and cannot be

changed, which indicates that the Timer Driver is a single-client driver). The last implementation-specific configuration option in this example is the "Interrupt Mode" (DRV_TMR_INTERRUPT_MODE) setting. This option determines if the implementation is configured to run polled or interrupt driven (discussed further, in a following section). MPLAB Harmony drivers are generally designed to run most effectively in an interrupt-driven configuration, but they can also be run in a polled configuration to simplify debugging or to support task prioritization in an RTOS configuration.

The remaining configuration options are all instance-specific initialization options. For a dynamic implementation of a driver, these options are passed into the driver's Initialize function through an "init" data structure, as shown in the following example.

Example Driver Init Structure in system_init.c

```
const DRV_TMR_INIT drvTmr0InitData =
{
    .moduleInit.sys.powerState = DRV_TMR_POWER_STATE_IDX0,
    .tmrId = DRV_TMR_PERIPHERAL_ID_IDX0,
    .clockSource = DRV_TMR_CLOCK_SOURCE_IDX0,
    .prescale = DRV_TMR_PRESCALE_IDX0,
    .mode = DRV_TMR_OPERATION_MODE_16_BIT,
    .interruptSource = DRV_TMR_INTERRUPT_SOURCE_IDX0,
    .asyncWriteEnable = false,
};
```

The exact meaning and usage of these options are described in the **Configuring the Library** section in the Help documentation for each library. The live MHC Help windowpane displays the associated help section whenever you select one of these options in the options tree.

There is one instance-specific initialization option of which you should take special notice: the peripheral ID option (.tmrId, in the Timer Driver example shown). This initialization option associates the driver instance (a zero-based index number) with the peripheral-hardware instance number, as defined by the data sheet for the processor in use. For a dynamic driver, this association is actually made when the driver's initialize function is called and passes a pointer to the init data structure, as shown in the following code example.

Example Driver Initialize Call in system_init.c

```
/* Initialize Drivers */
sysObj.drvTmr0 = DRV_TMR_Initialize(DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&drvTmr0InitData);
```

In this example, the driver index (DRV_TMR_INDEX_0) is defined as a numeric constant with a value of zero (0). This line of code associates driver instance 0 with hardware timer instance 1 by calling the DRV_TMR_Initialize function from the system initialization code and passing a pointer to the `drvTmr0InitData` structure. As shown earlier, the Timer Driver's init structure contains the value TMR_ID_1 (defined by the timer peripheral library), in its `.tmrId` data member.

In a static implementation, the driver peripheral ID macro (DRV_TMR_PERIPHERAL_ID_IDX0) defined in `system_config.h` is hard-coded into the driver's instance-specific initialization function when it is generated by the MHC, instead of defining an "init" structure, as shown in the following example; however, the effect is the same.

Example Static Driver Initialize Function

```
void DRV_TMR0_Initialize(void)
{
    PLIB_TMR_Stop(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_ClockSourceSelect(DRV_TMR_PERIPHERAL_ID_IDX0, DRV_TMR_CLOCK_SOURCE_IDX0);
    PLIB_TMR_PrescaleSelect(DRV_TMR_PERIPHERAL_ID_IDX0, DRV_TMR_PRESCALE_IDX0);
    PLIB_TMR_Mode16BitEnable(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_Counter16BitClear(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_Period16BitSet(DRV_TMR_PERIPHERAL_ID_IDX0, 0);
}
```

The DRV_TMR0_Initialize function (with an instance number '0' in the name) in the previous example, is a static version of the DRV_TMR_Initialize system interface function. The call to this function is created by the MHC when it generates the system code. Therefore, that call is always generated with the correct name and with the correct instance number in the name. However, when calling client interface functions (open, close, read, write, etc.) from your own applications, you *should not* use an instance number in the function name. Dynamic drivers implement the client interface functions without any index numbers in their names. Instead, they use an index or handle parameter to identify the instance of the driver with which to interact. Also, when using static implementations of the drivers, the dynamic API functions are mapped (using the index or handle parameter) to the appropriate static function with the index number in its name. Therefore, calling the dynamic API function makes your application always portable, using whichever driver instance is configured to the index value with which you open the driver.

Note: Calling the static versions of the interface function (with the index numbers in their names) is not prohibited. However, it will limit the portability of your application.

Understanding this mechanism is critical to understanding how to access the desired peripheral hardware instance. Therefore, it is worth looking at a few demonstration applications to see how it is used. Also, refer to *Volume IV: MPLAB Harmony Development > Key Concepts > Key One-to-Many Relationships* for additional information on the concepts of having multiple implementations, instances, and clients.

Something else worth noting about the previous example call to the Timer Driver's initialize functions is that when using a dynamic implementation, it returns a value called an "object handle". In the previous example, that object handle was stored in a system configuration object data member (`sysObj.drvTmr0`). Object handles returned by module initialization functions are stored in a system configuration structure normally named `sysObj`. The definition of this structure is generated in the `system_definitions.h` header file the MHC, as shown in the following example.

Example System Object Data Structure Definition in system_definitions.h

```
typedef struct
{
    SYS_MODULE_OBJ  sysDevcon;
```

```
    SYS_MODULE_OBJ  drvTmr0;

} SYSTEM_OBJECTS;

extern SYSTEM_OBJECTS sysObj;
```

As shown in the previous example, this structure is “extern’d” for use by the other system files. It should not be used by application or library files, only by the system files for a single configuration. The `sysObj` structure is defined (and allocated in memory) by the `system_init.c` file, as shown in the following example.

Example System sysObj Definition in system_init.c

```
/* Structure to hold the object handles for the modules in the system. */
SYSTEM_OBJECTS sysObj;
```

For this discussion, you can ignore the `sysDevcon` member of the SYSTEM_OBJECTS structure as it will contain the handle for a different library. The important thing to note is that the `drvTmr0` member must be passed into the Timer Driver’s other system interface functions so that the driver has access to the data it needs manage that specific instance of itself (and the associated peripheral hardware), as shown by the following timer ISR example.

Example Timer ISR in system_interrupt.c

```
void __ISR(_TIMER_1_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance0(void)
{
    DRV_TMR_Tasks(sysObj.drvTmr0);
}
```

In this ISR example, there are three important things to notice.

First, the ISR function itself is associated with a specific vector through the `__ISR` macro. Different interrupt vectors are associated with different peripheral instances and interrupts on different processors. That is why MPLAB Harmony ISR vector functions are generated in the configuration-specific `system_interrupt.c` file instead of being part of the driver library itself.

Second, the DRV_TMR_Tasks function implements the actual ISR logic of the TMR driver. Most MPLAB Harmony drivers are designed to run interrupt driven and their tasks functions implement the software state machine logic necessary to keep the driver’s interrupt sequence moving from one interrupt to the next until the driver’s task is complete.

Third, the `sysObj.drvTmr0` object handle’s value is passed into the driver’s tasks function so that it has access to the data it requires to control instance zero (0) of the Timer Driver and its associated hardware instance, which must match the ISR vector instance from which it is called.

By default, the Timer Driver is configured to run interrupt-driven, as shown previously. This is not necessarily true for all drivers. However, most drivers (including the Timer Driver) can run in a Polled mode by simply changing the configuration settings. For example, by clearing the "Interrupt Mode" option in the MHC configuration tree and regenerating the configuration code, the previous example ISR will be removed from `system_interrupt.c` and a call to the Timer Driver’s tasks function will be added to the polled system tasks function, as shown by the following `system_tasks.c` example code.

Example Call to Timer Tasks from system_tasks.c

```
void SYS_Tasks ( void )
{
    /* Maintain system services */
    SYS_DEVCON_Tasks(sysObj.sysDevcon);

    /* Maintain Device Drivers */
    DRV_TMR_Tasks(sysObj.drvTmr0);

    /* Maintain the application's state machine. */
    APP_Tasks();
}
```

In this example, the Timer Driver’s tasks function is called from the polled loop in main by the SYS_Tasks function. The driver’s tasks must still receive the `sysObj.drvTmr0` object handle value and its logic operates in exactly the same way, with one exception. Because the driver is now polled, the DRV_TMR_INTERRUPT_MODE option is now defined as false. This causes the driver to be built so that it does not enable its own interrupt, allowing it to run in the polled loop and to not require an ISR.

For additional information on the device driver system interface, refer to *Volume IV: MPLAB Harmony Development > MPLAB Harmony Driver Development Guide > System Interface* and to the documentation for the individual system interface functions for the driver in question.

Using a Driver's Client Interface

Introduces the Client Interface (or API) of a MPLAB Harmony device driver and explains common usage models.

Description

Applications (or any other “client” of a MPLAB Harmony device driver) normally only interact with the driver’s client interface (often called its API). The client interface provides functions to “open” the driver (creating a link between the client and the driver) and interact with it, to transfer data or perform operations that are specific to a given type of device, and to “close” the driver (releasing the link). Once a driver has been configured and the configuration code has been generated, the application can assume that the driver will be initialized by the system-wide initialization function (SYS_Initialize) and that its tasks functions will be called as required from either the system-wide tasks function (SYS_Tasks) or from the appropriate ISR, depending upon how the driver was designed and configured.

To interact with the driver, a client must first call the driver's open function. This is necessary because all other client interface functions require a "handle" to the device driver that is returned by the open function, as shown in the following example.

Example Call to a Driver's Open Function

```
appData.handleTmr = DRV_TMR_Open(APP_TMR_DRV_INDEX, DRV_IO_INTENT_EXCLUSIVE);
if( DRV_HANDLE_INVALID != appData.handleTmr )
{
    // Advance to next application state.
}
```

In this example, the first parameter to the DRV_TMR_Open function is the APP_TMR_DRV_INDEX macro, which is a constant defined to the value of the desired driver instance index number in the `system_config.h` header file. This value must be the same as the index number used when the desired driver was initialized (as shown in the previous section). This is how the client becomes associated with a specific instance of a driver.

The second parameter identifies how the client intends to use the driver. Here, the client wants to have exclusive access to the driver. This means that no other client can currently have an active handle to this driver or this call will fail and return a value of DRV_HANDLE_INVALID. Drivers can also be opened as shared, as blocking or non-blocking and for reading, writing, or both. Refer to the help for the DRV_IO_INTENT data type for additional information about the IO intent parameter of driver open functions. This parameter is merely an advisory parameter. How it is used by the driver is implementation dependent and will be described in the driver's help documentation.

Finally, if the open function was successful, the returned value will be a valid handle to the driver instance. This value is opaque and meaningless to the caller, but it must be passed back to the driver as the first parameter to every other client interface function provided by the driver. A valid handle identifies both the instance of the driver with which the caller interacts and it identifies the client performing the call. This means that, two different client applications or modules opening the same driver in the same system at the same time will receive different values for their "opened" handle. If, for any reason, the driver cannot support the "open" request (it is not finished initializing itself, it has already been opened for exclusive access, or cannot accept new open requests for any reason), it will return a value of DRV_HANDLE_INVALID, indicating the client cannot use it at this time. The DRV_HANDLE_INVALID value is the only non-opaque value that a client should consider meaningful. All other values are only meaningful to the driver that provided them.

Note: The `appData.handleTmr` variable in the previous example is a member of the application's `appData` structure. This structure is generated by the MHC as part of the initial application template and should be used to hold an applications state variables.

When the client is finished using a driver, it may close it, as shown in the following example.

Example Call to a Driver's Close Function

```
DRV_TMR_Close(appData.handleTmr);
```

This action releases the link to the driver, invalidating the handle and releasing any resources allocated by the driver to track requests from the client. Notice that the close function demonstrates the use of the driver handle, requiring it as a parameter. However, after the close function returns, the handle value cannot be used again. Therefore, the client should not call the driver's close function until it is done using the driver or it will have to call open again and obtain a new handle to use the driver again. In fact, since many embedded applications are always running, they often do not bother to close drivers they use. But, applications that can go idle or that can be stopped and restarted or that need to share a driver with other clients, but want to conserve resources, or that want use the driver exclusively, can close a driver when they are finished with it for a time and reopen it later when needed. In fact, this is a good way to share a single-client driver, or a driver that supports exclusive access, allowing each client to open it and use it only when a valid handle is obtained.

Using a Driver in an Application

Describes how to write a state-machine based application that uses a MPLAB Harmony driver.

Description

MPLAB Harmony generally treats all software modules, including applications, as state machines that have an "initialize" function and a "tasks" function. In fact, when not using a RTOS, it essentially treats the entire system as one large state machine that runs in a common super loop in the "main" function, as shown in the following code example.

Example Main Function

```
int main ( void )
{
    SYS_Initialize(NULL);

    while(true)
    {
        SYS_Tasks();
    }

    return (EXIT_FAILURE);
}
```

For the purpose of this discussion, it is important to understand that the application's APP_Initialize function is called from the SYS_Initialize function, along with the initialization of functions of all drivers and other libraries before execution enters the endless `while(true)` super loop that continuously calls the system-wide SYS_Tasks function. The application's APP_Tasks function is then called from the SYS_Tasks function inside of the super loop, along with all other polled modules in the system. If you are not already familiar with the organization of an MPLAB

Harmony project, please refer to *Volume I: Getting Started With MPLAB Harmony > What is MPLAB Harmony?* for more information.

An application that uses a driver must define a DRV_HANDLE variable, as shown in the following example application header file.

Example Driver Application Header (app.h)

```
#include "driver/usart/drv_usart.h"

typedef enum
{
    APP_STATE_SETUP=0,
    APP_STATE_MESSAGE_SEND,
    APP_STATE_MESSAGE_WAIT,
    APP_STATE_DONE

} APP_STATES;


typedef struct
{
    APP_STATES  state;
    DRV_HANDLE  usart;
    char *      message;

} APP_DATA;
```

In this previous example, the driver handle variable is named `usart`. To keep the application well organized, it is common to keep all of the application's state variables (including one called "state" that holds the current state of the application's state machine) in a common structure (APP_DATA). This structure must be allocated in the application's source file (usually named `app.c`) and initialized by the application's initialization function, as shown in the following example.

Example Driver Application Initialization

```
APP_DATA appData;

void APP_Initialize ( void )
{
    /* Place the App in its initial state. */
    appData.state   = APP_STATE_SETUP;
    appData.usart   = DRV_HANDLE_INVALID;
    appData.message = "Hello World\n";
}
```

The APP_Initialze function must initialize the state variable (appData.state) to put the application's state machine in its initial state (the APP_STATE_SETUP value from the APP_STATES enumeration). It must also initialize the driver-handle variable (`appData.usart`), so that the state machine knows it is not yet valid, and any other application variables (like the string pointer, `appData.message`).

Once the application's data structure has been initialized, it is safe for the system (the main and SYS_Tasks functions) to call the application's APP_Tasks function from the super loop to keep it running. The APP_Tasks function then executes state transition code as it switches between states, as demonstrated by the following example.

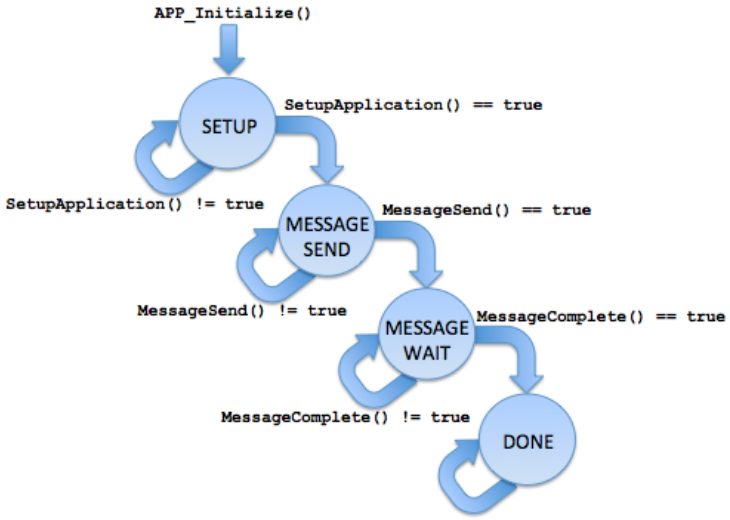
Example Application State Machine Using a Driver

```
void APP_Tasks ( void )
{
    switch ( appData.state )
    {
        case APP_STATE_SETUP:
        {
            if (SetupApplication() == true)
            {
                appData.state = APP_STATE_MESSAGE_SEND;
            }
            break;
        }

        case APP_STATE_MESSAGE_SEND:
        {
            if (MessageSend() == true)
            {
                appData.state = APP_STATE_MESSAGE_WAIT;
            }
            break;
        }

        case APP_STATE_MESSAGE_WAIT:
        {
```

```
            if (MessageComplete() == true)
            {
                appData.state = APP_STATE_DONE;
            }
            break;
        }

        case APP_STATE_DONE:
        default:
        {
            break;
        }
    }
}
```

There are numerous ways to implement a state machine. However, in this example, the application changes state when the APP_Tasks function assigns a new value from the APP_STATES enumeration to the `appData.states` variable. This happens when one of the state transition function returns true. The end result is an overall application state machine execution that retries each state transition until it succeeds before moving on to the next state, as shown in the following diagram.

Application State Machine



Note: The APP_STATE_ prefix and all inter-word underscores were removed from the state names to simplify the diagram.

After APP_Initialize places the state machine in its initial APP_STATE_SETUP state, the APP_Tasks function will call the SetupApplication function when it is called. When SetupApplication returns true indicating it has completed its task, the state machine advances to the next state. Otherwise, it stays in the same state and retries the tasks in the SetupApplication function. This pattern repeats for the APP_STATE_MESSAGE_SEND state and the `MessageSend` function as well as the APP_STATE_MESSAGE_WAIT state and the `MessageComplete` function. When all functions have returned true, the state machine to transitions to the APP_STATE_DONE state where it unconditionally stays having completed its tasks.

The sum total of the tasks performed by each transition function completes the overall task of the application. For an application that uses a driver like this example, this includes opening the driver, sending the message, and closing the driver when the message has been sent. How each individual transition function in this example application accomplishes its portion of the overall task, is described in the examples in the following sections to demonstrate how drivers are commonly used.

Opening a Driver

Describes how to open a driver in a state-machine based application.

Description

To use a MPLAB Harmony driver, an application (or other client) must call the driver's "open" function and obtain a valid handle to it, as shown by the following code example.

Example Opening a Driver

```
static bool SetupApplication ( void )
{
    if (appData.usart == DRV_HANDLE_INVALID)
    {
```

```
        appData.usart = DRV_USART_Open(APP_USART_DRIVER_INDEX,
                                       (DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING));
    }

    if (appData.usart == DRV_HANDLE_INVALID)
    {
        return false;
    }

    return true;
}
```

This example demonstrates the implementation of a state-transition function in a state machine-based application (as shown in the previous Using a Driver in an Application section). The `SetupApplication` function assumes that the `appData.usart` variable has been initialized to a value of DRV_HANDLE_INVALID when the application's state machine was initialized. Therefore, it checks this variable every time it is called to see if it has already completed its task. If `appData.usart` contains a value of DRV_HANDLE_INVALID, this indicates that the driver has not yet been successfully opened, causing the function to attempt to open the driver by calling DRV_USART_Open.

If the USART driver is ready and able to support a new client it will return a valid handle. If it is not ready or able to accept a new client, the driver will return DRV_HANDLE_INVALID and the `SetupApplication` function will return false and the application will stay in the same state and try to open the driver again the next time its state machine tasks function is called. When DRV_USART_Open returns a valid handle (a handle that is not equal to DRV_HANDLE_INVALID), the `SetupApplication` function returns true, allowing the application's state machine to advance.

This technique allows the application to try repeatedly to open the driver until it succeeds and guarantees that the application's state machine will not advance until it has done so. A more sophisticated application might use a time-out mechanism or some other error handling logic to take alternative action if it cannot open the driver in an acceptable time period. However, this simple implementation demonstrates the basic concept of how an MPLAB Harmony application (or any other client module) can safely open a driver before attempting to use it.

Using Driver Interface Functions

Describes how to use a device driver's synchronous client interface functions, such as those that read and write data.

Description

To use a MPLAB Harmony driver's client interface, the application must first obtain a valid handle from the driver's "open" function. The examples in this section assume that that has already occurred and that the value of the USART driver handle in the appData.usart variable is valid. The following example code demonstrates the implementation of a state transition function in a state machine-based application (as shown in the previous Using a Driver in an Application section) that writes data to a USART driver for transmission on the associated USART peripheral.

Example Writing Data To a Driver

```
static bool MessageSend ( void )
{
    size_t  count;
    size_t  length = strlen(appData.message);

    count = DRV_USART_Write(appData.usart, appData.message, length);

    appData.message += count;

    if (count == length)
    {
        return true;
    }

    return false;
}
```

In this example, the appData.message variable is a char pointer pointing to a null-terminated C-language string that was defined and initialized, as shown in the Using a Driver in an Application section. When `MessageSend` function is first called by the application's state machine, it points to the first character in the string to be transmitted. The function calculates the current length of the message string (using the standard C-language strlen function) and calls the driver's DRV_USART_Write function, passing it the valid driver handle (`appData.usart`) along with the pointer to the message string and its length, to transmit the message string on the associated USART.

If the driver is configured for blocking, the DRV_USART_Write function will not return until it has processed all of the data in the message string. However, that usually requires the use of a RTOS. Normally, in a bare-metal system (one that does not use a RTOS), MPLAB Harmony drivers are used in a non-blocking mode. In that case, a driver will perform as much of a task as it can when one of its interface functions is called without blocking. This means that the function will then return immediately, not waiting for the task to complete, and provide information on how much of the task was completed so the client can react appropriately. In this example, the DRV_USART_Write function will return a count of the number of bytes that were processed by the USART driver by this call to the function.

The `MessageSend` function captures the number of bytes processed by the DRV_USART_Write function in a local count variable. It then effectively removes those bytes from the message string by incrementing the pointer by count bytes (`appData.message` is a char pointer that increments by the size of one byte for every '1' added to it). Then, the `MessageSend` function checks to see if it was able to write the entire string by comparing the value of count to the value of length that it calculated before calling the driver's write function. If the two are equal, the task is

complete and the MessageSend function returns true and the application's state machine can continue to the next state. If the two values are not equal, this indicates there are remaining bytes in the message string. The MessageSend function returns false and the application must stay in the same state so that the function can attempt to send the remaining bytes next time it is called. A driver only accepts data when it can process it; therefore, the client can call its data transfer function as many times as necessary, even when the function returns bytes processed if it cannot accept more data at that time.

When a client has called a driver interface function there are really only two possibilities. Either the operation has completed when the function returns, or the operation continues after the function has returned. If the operation completes immediately, the client can continue on without taking further action. However, in this example, while the USART driver may have accepted some of the bytes in the message string (perhaps copying them to an internal hardware or software FIFO buffer), it still takes some time to transmit the data over the USART peripheral. In many cases the client may need to know when the operation has actually completed. For this reason, most drivers provide one or more status functions that client applications may call to determine the current status of an operation, as demonstrated in the following example.

Example Using a Driver Status Function

```
static bool MessageComplete ( void )
{
    if (DRV_USART_ClientStatus(appData.usart) == DRV_USART_CLIENT_STATUS_BUSY)
    {
        return false;
    }
    return true;
}
```

This example extends the previous one and assumes that the MessageSend function has returned true and the application has moved to a new state where it calls this function to determine when the driver is idle, which indicates that the message has been completely transmitted. To do that, the MessageComplete function calls the DRV_USART_ClientStatus function. If its return value is DRV_USART_CLIENT_STATUS_BUSY, the USART driver is still working on a previous request by the client. If any other status value is returned, this indicates that the driver is no longer busy with a current request and the MessageComplete function returns true so that the client application's state machine can move on. A more sophisticated example would check for other possible status values that might indicate some error has occurred and take appropriate action. However, this example is sufficient to demonstrate the concept of checking a driver status function to determine when it is safe to move to another state.

Since the client application stays in the same state calling the status function each time its tasks function is called until the desired status is returned, it is effectively polling the status as if it were in a while loop. In fact, it is in the system-wide while loop. However, by not trapping the CPU within its own internal while loop, the application allows other modules (including, potentially, the driver it is using) to continue running and servicing requests. Failing to allow the rest of the system to run can result in a deadlock where the polling application is waiting for a status; however, the driver it is polling will never be able to provide the expected status, as the driver's own tasks function is not allowed to run. This is why it is important to use the technique described here to "poll" status from modules outside of the current module.

Using Asynchronous and Callback Functions

Describes how to use an asynchronous interface function to start a driver operation and receive a callback when the operation is complete.

Description

When a client calls a function that is part of an asynchronous interface, the function starts the request and returns immediately, without finishing the request. The client can then either poll a status function to determine when the request has finished (as demonstrated in the Using Driver Interface Functions section) or it can utilize a callback function to receive a notification from the driver when the request has finished. So, the difference between an asynchronous interface and a synchronous interface is that a synchronous interface may finish all or part of the request before returning, whereas an asynchronous interface will always return immediately having only started the request. Determination of when the request has completed is handled separately.

The examples in this section reimplement some of the code from the example application described in the previous sections to demonstrate how to use asynchronous queuing and callback interfaces instead of the synchronous status-polling interface demonstrated in the Using Driver Interface Functions section. To use an asynchronous interface, we will first add a couple of new variables to our example application's data structure, as shown by the following structure definition.

Example Driver Application Header (app.h)

```
typedef struct
{
    APP_STATES              state;
    DRV_HANDLE              usart;
    char *                  message;
    DRV_USART_BUFFER_HANDLE messageHandle;
    bool                    messageDone;

} APP_DATA;
```

The state, usart, and message members of the APP_DATA structure are used in exactly the same way as they were in the previous examples. The messageHandle variable will be explained later and the messageDone variable is a Boolean flag used by the callback function to indicate to the application's state machine that the message has been completely processed by the driver. Using these new mechanisms results in very minor changes to the application's state machine, as shown in the following example APP_Initialize and APP_Tasks implementations.

Example Driver Application State Machine (app.c)

```
void APP_Initialize ( void )
{
    appData.state           = APP_STATE_SETUP;
    appData.usart           = DRV_HANDLE_INVALID;
    appData.message         = APP_MESSAGE;
    appData.messageHandle   = DRV_USART_BUFFER_HANDLE_INVALID;
}

void APP_Tasks ( void )
{
    switch ( appData.state )
    {
        case APP_STATE_SETUP:
        {
            if (SetupApplication() == true)
            {
                appData.state = APP_STATE_MESSAGE_SEND;
            }
            break;
        }

        case APP_STATE_MESSAGE_SEND:
        {
            if (MessageSend() == true)
            {
                appData.state = APP_STATE_MESSAGE_WAIT;
            }
            break;
        }

        case APP_STATE_MESSAGE_WAIT:
        {
            if (appData.messageDone)
            {
                DRV_USART_Close(appData.usart);
                appData.state = APP_STATE_DONE;
            }
            break;
        }

        case APP_STATE_DONE:
        default:
        {
            break;
        }
    }
}
```

As described previously, the `SetupApplication` state transition function opens the USART driver and the `MessageSend` function sends the message to it. However, there is no need for a `MessageComplete` state transition function. Instead, the application must implement a callback function that will set the `appData.messageDone` Boolean flag when the driver calls the application "back" to indicate that the message has been sent.

Note: The `AppInitialize` function initializes the `state`, `usart`, and `message` members of the `appData` structure as previously described. And, it also initializes the `messageHandle` member with an invalid value to indicate that the message has not yet been sent. However, it does not initialize the `messageDone` flag because it is more appropriate to clear the flag elsewhere, immediately before calling the driver to send the message.

To use a callback mechanism requires the client to implement and register a callback function. A client must register this function after opening the driver, but prior to calling the driver to initiate the operation. This is often done in the same state transition that opens the driver, as shown in the following `SetupApplication` example.

Example Registering a Driver Callback Function

```
static void BufferDone ( DRV_USART_BUFFER_EVENT  event,
                         DRV_USART_BUFFER_HANDLE bufferHandle,
                         uintptr_t context )
{
    APP_DATA *pAppData = (APP_DATA *)context;

    if (event == DRV_USART_BUFFER_EVENT_COMPLETE)
```

```
    {
        if (bufferHandle == pAppData->messageHandle)
        {
            pAppData->messageDone = true;
            return;
        }
    }

    /* Error */
    return;
}


static bool SetupApplication ( void )
{
    if (appData.usart == DRV_HANDLE_INVALID)
    {
        appData.usart = DRV_USART_Open(APP_USART_DRIVER_INDEX,
                                       (DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING));
    }

    if (appData.usart == DRV_HANDLE_INVALID)
    {
        return false;
    }

    DRV_USART_BufferEventHandlerSet(appData.usart, BufferDone, (uintptr_t)&appData);
    return true;
}
```

This code block implements both the `BufferDone` callback function and the application's `SetupApplication` state transition function. After successfully opening the driver, the `SetupApplication` function calls the DRV_USART_BufferEventHandlerSet function and passes it the driver handle (`appData.usart`) once it is valid, along with the address of the `BufferDone` callback function and a context value.

The context value can be anything that will fit in an integer large enough to hold a pointer (it is a `uintptr_t` variable). However, this parameter is most commonly used to pass a pointer to the caller's own data structure as demonstrated here (even though it is not strictly necessary). This is done primarily to support multi-instance clients. (Refer to *Volume IV: MPLAB Harmony Development > Key Concepts* for information on multiple instances.) A multi-instance client is designed to manage multiple instances of itself by allocating multiple instances of its own data structure, but only one instance of its object code. Passing a pointer to the data structure in the context variable identifies the specific instance that was used when calling the driver.

Once the callback function has been registered with the driver, the application can transition to a state where it attempts to initiate an asynchronous operation. The following example demonstrates the use of a buffer-queuing write function to transmit a message over the USART.

Example Queuing a Buffer to a Driver

```
static bool MessageSend ( void )
{
    appData.messageDone = false;
    DRV_USART_BufferAddWrite(appData.usart, &appData.messageHandle,
                             appData.message, strlen(appData.message));

    if (appData.messageHandle == DRV_USART_BUFFER_HANDLE_INVALID)
    {
        return false;
    }

    return true;
}
```

Before attempting to send the message, this implementation of the `MessageSend` state transition function clears the `appData.messageDone` flag so it can detect when the message has completed. Then, it calls the DRV_USART_BufferAddWrite function to queue up the buffer containing the message to be transmitted by the USART driver. To that function, it passes the USART driver handle (`appData.usart`), the address of the `appData.messageHandle` variable, the pointer to the message buffer (`appData.message`), and the size of the buffer in bytes as calculated by the `strlen` function. The USART driver then adds this buffer to its internal queue of buffers to transmit and provides a handle to the caller that identifies that buffer's place in the queue by storing it to the `appData.messageHandle` variable.

If, for some reason, the driver is unable to successfully queue up the buffer (perhaps the queue is full), it will assign a value of DRV_USART_BUFFER_HANDLE_INVALID to the `appData.messageHandle` variable. If that happens, the `MessageSend` function returns false and the application will stay in the same state and retry the operation again next time its tasks function is called. But, if the operation succeeds, the application advances to the next state.

Once the driver completes the operation, it will call the client's callback function. As shown in the `BufferDone` code example, the driver passes it an enumeration value that identifies which event has just occurred (the `DRV_USART_BUFFER_EVENT_COMPLETE` value) in the event parameter. It also passes it the handle of the buffer that has just completed (`bufferHandle`). The client can use the `bufferHandle` value to verify that it

matches the value stored in the appData.bufferHandle variable to uniquely identify an individual buffer. This is very useful when a client queues up multiple buffers at the same, which is being shown in this example as a demonstration.

The context parameter to the BufferDone function contains a pointer to the application's global (appData) data structure. (This is the same value that was passed in the context parameter to the DRV_USART_BufferEventHandlerSet function.) While not strictly necessary in this example, it is very useful for multi-instance clients such as dynamic device drivers and middleware to identify which instance of the client requested the operation. The callback function simply casts the context value back into a pointer to the client's own data structure's data type (APP_DATA in this example) and uses it to access the structure members. (Again, please refer to *Volume IV: MPLAB Harmony Development > Key Concepts* for information on multiple instances.)

The callback function uses the event parameter to identify why the callback occurred. If it was called to indicate that the buffer has been processed, the event parameter will contain the value DRV_USART_BUFFER_EVENT_COMPLETE. If it contains any other value an error has occurred. The BufferDone callback also checks to verify that the buffer that completed was the same buffer that it queued up by comparing the bufferHandle value it was passed with the value assigned to the appData.messageHandle variable when the application called DRV_USART_BufferAddWrite. It accesses the message handle value it saved using the pAppData pointer given to it through the context parameter just. Once it has verified that the buffer it queued has completed, it sets the pAppData->messageDone flag to notify the application's state machine and execution returns to the driver.

Note: It is important to understand that the MessageDone callback function executes in the context of the driver, not the application. Depending on how the system is configured, this means that it may be called from within the driver's ISR context or from another thread context if using a RTOS.

In this example, the APP_Tasks application state machine function is essentially the same as the state machine for the synchronous example. The only difference is that when the application is in the APP_STATE_MESSAGE_WAIT state, it checks the appData.messageDone flag to determine when to close the driver and transition to the APP_STATE_DONE state instead of calling a transition function. (It could still do this in a state transition function, but it was done differently in this example to emphasize the concept.)

The advantage of using an asynchronous interface over a synchronous one is that it allows the client's state machine to continue on, potentially doing something else while the requested operation completes. Whereas a synchronous interface has the possibility of blocking the client's state machine until the operation finishes (when used in a RTOS configuration). An asynchronous interface will always return immediately without blocking (whether a RTOS is used or not). Because of this, most asynchronous interfaces will also allow queuing of more than one operation at a time. This allows client applications to keep a driver continuously busy by keeping the driver's queue full, maximizing data throughput or operation speed. By contrast, a synchronous interface requires one operation to complete before the synchronous function can be called again to cause the next one to begin.

The cost of this capability is that an asynchronous interface has the added complexity of a callback function (if the client cares when the operation finishes) and the fact that a callback function may be called from within the driver's ISR context, depending on how the driver was designed and configured. This fact generally restricts what can be done within the callback function. For example, it is usually a bad idea to perform lengthy processing within a callback function as it will block all lower priority ISRs (as well as the main loop or other threads) while that processing occurs. Also, it is usually best to not call back into the driver's own interface functions unless those functions are documented as being safe to call from within the driver's callback context. Many interface functions (particularly data transfer and data queuing functions) must use semaphores or mutexes to protect their internal data structures in RTOS environments and those constructs cannot be used from within an ISR.

It is also important to not make non-atomic (read-modify-write) accesses to the client's own state data from within the callback function, as the client cannot protect itself against an interrupt that is owned by the driver. That is why a separate Boolean flag variable is commonly used to indicate to the client that the callback has occurred. Most other processing should occur in the client's state machine. It is usually best to simply capture the event and return as quickly as possible from the callback function and let the application's state machine tasks function perform any lengthy processing or calling back into the driver.

Please refer to *Volume IV: MPLAB Harmony Development* for additional information.

Library Interface

Constants

	Name	Description
	DRV_CONFIG_NOT_SUPPORTED	Not supported configuration.
	DRV_HANDLE_INVALID	Invalid device handle.
	DRV_IO_ISBLOCKING	Returns if the I/O intent provided is blocking
	DRV_IO_ISEXCLUSIVE	Returns if the I/O intent provided is non-blocking.
	DRV_IO_ISNONBLOCKING	Returns if the I/O intent provided is non-blocking.
	_DRV_COMMON_H	This is macro _DRV_COMMON_H.
	_PLIB_UNSUPPORTED	Abstracts the use of the unsupported attribute defined by the compiler.

Data Types

	Name	Description
	DRV_CLIENT_STATUS	Identifies the current status/state of a client's connection to a driver.
	DRV_HANDLE	Handle to an opened device driver.
	DRV_IO_BUFFER_TYPES	Identifies to which buffer a device operation will apply.

	DRV_IO_INTENT	Identifies the intended usage of the device when it is opened.

Description

Data Types

DRV_CLIENT_STATUS Enumeration

Identifies the current status/state of a client's connection to a driver.

File

driver_common.h

C

```
typedef enum {
  DRV_CLIENT_STATUS_ERROR_EXTENDED = -10,
  DRV_CLIENT_STATUS_ERROR = -1,
  DRV_CLIENT_STATUS_CLOSED = 0,
  DRV_CLIENT_STATUS_BUSY = 1,
  DRV_CLIENT_STATUS_READY = 2,
  DRV_CLIENT_STATUS_READY_EXTENDED = 10
} DRV_CLIENT_STATUS;
```

Members

Members	Description
DRV_CLIENT_STATUS_ERROR_EXTENDED = -10	Indicates that a driver-specific error has occurred.
DRV_CLIENT_STATUS_ERROR = -1	An unspecified error has occurred.
DRV_CLIENT_STATUS_CLOSED = 0	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_CLIENT_STATUS_BUSY = 1	The driver is currently busy and cannot start additional operations.
DRV_CLIENT_STATUS_READY = 2	The module is running and ready for additional operations
DRV_CLIENT_STATUS_READY_EXTENDED = 10	Indicates that the module is in a driver-specific ready/run state.

Description

Driver Client Status

This enumeration identifies the current status/state of a client's link to a driver.

Remarks

The enumeration used as the return type for the client-level status routines defined by each device driver or system module (for example, DRV_USART_ClientStatus) must be based on the values in this enumeration.

DRV_HANDLE Type

Handle to an opened device driver.

File

driver_common.h

C

```
typedef uintptr_t DRV_HANDLE;
```

Description

Device Handle

This handle identifies the open instance of a device driver. It must be passed to all other driver routines (except the initialization, deinitialization, or power routines) to identify the caller.

Remarks

Every application or module that wants to use a driver must first call the driver's open routine. This is the only routine that is absolutely required for

every driver.

If a driver is unable to allow an additional module to use it, it must then return the special value DRV_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

DRV_IO_BUFFER_TYPES Enumeration

Identifies to which buffer a device operation will apply.

File

driver_common.h

C

```
typedef enum {
  DRV_IO_BUFFER_TYPE_NONE = 0x00,
  DRV_IO_BUFFER_TYPE_READ = 0x01,
  DRV_IO_BUFFER_TYPE_WRITE = 0x02,
  DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ|DRV_IO_BUFFER_TYPE_WRITE
} DRV_IO_BUFFER_TYPES;
```

Members

Members	Description
DRV_IO_BUFFER_TYPE_NONE = 0x00	Operation does not apply to any buffer
DRV_IO_BUFFER_TYPE_READ = 0x01	Operation applies to read buffer
DRV_IO_BUFFER_TYPE_WRITE = 0x02	Operation applies to write buffer
DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ|DRV_IO_BUFFER_TYPE_WRITE	Operation applies to both read and write buffers

Description

Device Driver IO Buffer Identifier

This enumeration identifies to which buffer (read, write, both, or neither) a device operation will apply. This is used for "flush" (or similar) operations.

DRV_IO_INTENT Enumeration

Identifies the intended usage of the device when it is opened.

File

driver_common.h

C

```
typedef enum {
  DRV_IO_INTENT_READ,
  DRV_IO_INTENT_WRITE,
  DRV_IO_INTENT_READWRITE,
  DRV_IO_INTENT_BLOCKING,
  DRV_IO_INTENT_NONBLOCKING,
  DRV_IO_INTENT_EXCLUSIVE,
  DRV_IO_INTENT_SHARED
} DRV_IO_INTENT;
```

Members

Members	Description
DRV_IO_INTENT_READ	Read
DRV_IO_INTENT_WRITE	Write
DRV_IO_INTENT_READWRITE	Read and Write
DRV_IO_INTENT_BLOCKING	The driver will block and will return when the operation is complete
DRV_IO_INTENT_NONBLOCKING	The driver will return immediately
DRV_IO_INTENT_EXCLUSIVE	The driver will support only one client at a time
DRV_IO_INTENT_SHARED	The driver will support multiple clients at a time

Description

Device Driver I/O Intent

This enumeration identifies the intended usage of the device when the caller opens the device. It identifies the desired behavior of the device

driver for the following:

- Blocking or non-blocking I/O behavior (do I/O calls such as read and write block until the operation is finished or do they return immediately and require the caller to call another routine to check the status of the operation)
- Support reading and/or writing of data from/to the device
- Identify the buffering behavior (sometimes called "double buffering" of the driver. Indicates if the driver should maintain its own read/write buffers and copy data to/from these buffers to/from the caller's buffers.
- Identify the DMA behavior of the peripheral

Remarks

The buffer allocation method is not identified by this enumeration. Buffers can be allocated statically at build time, dynamically at run-time, or even allocated by the caller and passed to the driver for its own usage if a driver-specific routine is provided for such. This choice is left to the design of the individual driver and is considered part of its interface.

These values can be considered "flags". One selection from each of the groups below can be ORed together to create the complete value passed to the driver's open routine.

Constants

DRV_CONFIG_NOT_SUPPORTED Macro

Not supported configuration.

File

driver_common.h

C

```
#define DRV_CONFIG_NOT_SUPPORTED (((unsigned short) -1))
```

Description

Not supported configuration

If the configuration option is not supported on an instance of the peripheral, use this macro to equate to that configuration. This option should be listed as a possible value in the description of that configuration option.

DRV_HANDLE_INVALID Macro

Invalid device handle.

File

driver_common.h

C

```
#define DRV_HANDLE_INVALID (((DRV_HANDLE) -1))
```

Description

Invalid Device Handle

If a driver is unable to allow an additional module to use it, it must then return the special value DRV_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

Remarks

None.

DRV_IO_ISBLOCKING Macro

Returns if the I/O intent provided is blocking

File

driver_common.h

C

```
#define DRV_IO_ISBLOCKING(intent) (intent & DRV_IO_INTENT_BLOCKING)
```

Description

Device Driver Blocking Status Macro

This macro returns if the I/O intent provided is blocking.

Remarks

None.

DRV_IO_ISEXCLUSIVE Macro

Returns if the I/O intent provided is non-blocking.

File

driver_common.h

C

```
#define DRV_IO_ISEXCLUSIVE(intent) (intent & DRV_IO_INTENT_EXCLUSIVE)
```

Description

Device Driver Exclusive Status Macro

This macro returns if the I/O intent provided is non-blocking.

Remarks

None.

DRV_IO_ISNONBLOCKING Macro

Returns if the I/O intent provided is non-blocking.

File

driver_common.h

C

```
#define DRV_IO_ISNONBLOCKING(intent) (intent & DRV_IO_INTENT_NONBLOCKING )
```

Description

Device Driver Non Blocking Status Macro

This macro returns if the I/ intent provided is non-blocking.

Remarks

None.

_DRV_COMMON_H Macro

File

driver_common.h

C

```
#define _DRV_COMMON_H
```

Description

This is macro _DRV_COMMON_H.

_PLIB_UNSUPPORTED Macro

Abstracts the use of the unsupported attribute defined by the compiler.

File

driver_common.h

C

```
#define _PLIB_UNSUPPORTED
```

Description

Unsupported Attribute Abstraction

This macro nulls the definition of the _PLIB_UNSUPPORTED macro, to support compilation of the drivers for all different variants.

Remarks

None.

Example

void _PLIB_UNSUPPORTED PLIB_USART_Enable(USART_MODULE_ID index);

This function will not generate a compiler error if the interface is not defined for the selected device.

Files

Files

Name	Description
driver.h	This file aggregates all of the driver library interface headers.
driver_common.h	This file defines the common macros and definitions used by the driver definition and implementation headers.

Description

driver.h

This file aggregates all of the driver library interface headers.

Description

Driver Library Interface Header Definitions

Driver Library Interface Header This file aggregates all of the driver library interface headers so client code only needs to include this one single header to obtain prototypes and definitions for the interfaces to all driver libraries. A device driver provides a simple well-defined interface to a hardware peripheral that can be used without operating system support or that can be easily ported to a variety of operating systems. A driver has the fundamental responsibilities:

- Providing a highly abstracted interface to a peripheral
- Controlling access to a peripheral
- Managing the state of a peripheral

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

drv.h

Company

Microchip Technology Inc.

driver_common.h

This file defines the common macros and definitions used by the driver definition and implementation headers.

Enumerations

	Name	Description
	DRV_CLIENT_STATUS	Identifies the current status/state of a client's connection to a driver.
	DRV_IO_BUFFER_TYPES	Identifies to which buffer a device operation will apply.
	DRV_IO_INTENT	Identifies the intended usage of the device when it is opened.

Macros

	Name	Description
	_DRV_COMMON_H	This is macro _DRV_COMMON_H.
	_PLIB_UNSUPPORTED	Abstracts the use of the unsupported attribute defined by the compiler.
	DRV_CONFIG_NOT_SUPPORTED	Not supported configuration.
	DRV_HANDLE_INVALID	Invalid device handle.
	DRV_IO_ISBLOCKING	Returns if the I/O intent provided is blocking
	DRV_IO_ISEXCLUSIVE	Returns if the I/O intent provided is non-blocking.
	DRV_IO_ISNONBLOCKING	Returns if the I/O intent provided is non-blocking.

Types

	Name	Description
	DRV_HANDLE	Handle to an opened device driver.

Description

Driver Common Header Definitions

This file defines the common macros and definitions used by the driver definition and the implementation header.

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

drv_common.h

Company

Microchip Technology Inc.

ADC Driver Library

This section describes the Analog-to-Digital Converter (ADC) Driver Library.

Introduction

This Analog-to-Digital Converter (ADC) driver provides an interface to manage the ADC module on the Microchip family of microcontrollers.

Description

An ADC is a vital part of any system that interfaces to real-world signals. While there are many techniques for analog-to-digital conversion, the Microchip family of microcontrollers uses Successive Approximation as one of its primary techniques.

Through MHC, this driver provides APIs to interact with the ADC module.

Note: Only Static implementation is supported for the ADC Driver Library.

Library Interface

Functions

	Name	Description
	DRV_ADC_Deinitialize	Deinitializes the DRV_ADC_Initialize driver module **Implementation:** Static
	DRV_ADC_Initialize	Initializes the ADC driver. **Implementation:** Static
	DRV_ADC_SamplesAvailable	Identifies if specified ADC Driver input has any samples available to read. **Implementation:** Static
	DRV_ADC_SamplesRead	Reads the converted sample data from ADC input Data buffer. **Implementation:** Static
	DRV_ADC_Start	Starts the software trigger for the ADC driver sampling and converting analog to digital values. **Implementation:** Static

	DRV_ADC_Stop	Stops the Global Software Level Trigger from continuing triggering for converting ADC data. **Implementation:** Static
	DRV_ADCx_Close	Closes the ADC instance for the specified driver index. **Implementation:** Static
	DRV_ADCx_Open	Opens the ADC instance for the specified driver index. **Implementation:** Static

Description

This section lists the interface routines, data types, constants and macros for the library.

Functions

DRV_ADC_Deinitialize Function

Deinitializes the DRV_ADC_Initialize driver module

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Deinitialize();
```

Returns

None.

Description

This function deinitializes the ADC Driver module for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

None.

Function

void DRV_ADC_Deinitialize(void)

DRV_ADC_Initialize Function

Initializes the ADC driver.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Initialize();
```

Returns

None.

Description

This function initializes the ADC Driver module for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This function must be called before any other ADC function is called. This function should only be called once during system initialization.

Preconditions

None.

Function

void DRV_ADC_Initialize(void)

DRV_ADC_SamplesAvailable Function

Identifies if specified ADC Driver input has any samples available to read.

Implementation: Static

File

help_drv_adc.h

C

```
bool DRV_ADC_SamplesAvailable(uint8_t bufIndex);
```

Returns

- true - When ADC data buffer is available to be read
- false - When ADC data buffer is not available

Description

This function identifies whether the specified ADC Driver input has any samples available to read.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open
- DRV_ADC_Start or other triggered by source setup in MHC

Parameters

Parameters	Description
uint8_t bufIndex	ADC input number (ANx)

Function

bool DRV_ADC_SamplesAvailable(uint8_t bufIndex);

DRV_ADC_SamplesRead Function

Reads the converted sample data from ADC input Data buffer.

Implementation: Static

File

help_drv_adc.h

C

```
uint32_t DRV_ADC_SamplesRead(uint8_t bufIndex);
```

Returns

uint32_t - ADC converted sample data.

Description

This function returns the converted sample data from ADC input Data buffer.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open
- DRV_ADC_Start or other triggered by source setup in MHC

Parameters

Parameters	Description
uint8_t bufIndex	Analog input number (ANx)

Function

uint32_t DRV_ADC_SamplesRead(uint8_t bufIndex);

DRV_ADC_Start Function

Starts the software trigger for the ADC driver sampling and converting analog to digital values.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Start();
```

Returns

None.

Description

This function provides a global edge and level trigger for the ADC driver to start the conversion.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open

Function

void DRV_ADC_Start(void);

DRV_ADC_Stop Function

Stops the Global Software Level Trigger from continuing triggering for converting ADC data.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Stop();
```

Returns

None.

Description

This function stops the Global Software Level Trigger from continuing triggering for converting ADC data.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open

Function

void DRV_ADC_Stop(void);

DRV_ADCx_Close Function

Closes the ADC instance for the specified driver index.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADCx_Close();
```

Returns

None.

Description

This function closes the specified driver instance (where 'x' is the instance number) making it ready for clients to use it.

Remarks

'x' indicates the instance number.

Preconditions

DRV_ADC_Initialize has been called.

Function

void DRV_ADCx_Close(void)

DRV_ADCx_Open Function

Opens the ADC instance for the specified driver index.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADCx_Open();
```

Returns

None.

Description

This function opens the specified driver instance (where 'x' is the instance number) making it ready for clients to use it.

Remarks

'x' indicates the instance number.

Preconditions

DRV_ADC_Initialize has been called.

Function

void DRV_ADCx_Open(void)

Bluetooth Driver Libraries

This section describes the Bluetooth Driver Libraries that are included in your installation of MPLAB Harmony.

BM64 Bluetooth Driver Library

This section describes the BM64 Bluetooth Driver Library.

Introduction

This library provides an Applications Programming Interface (API) to manage a BM64 Module that is connected to a Microchip PIC32 microcontroller using UART and I²S for providing Bluetooth solutions for audio and Bluetooth Low Energy (BLE) applications.

Description

The BM64 is a Bluetooth 4.2 Stereo Module that supports classic A2DP, AVRCP, HFP, HSP, and SPP protocols, as well as Bluetooth Low Energy (BLE).

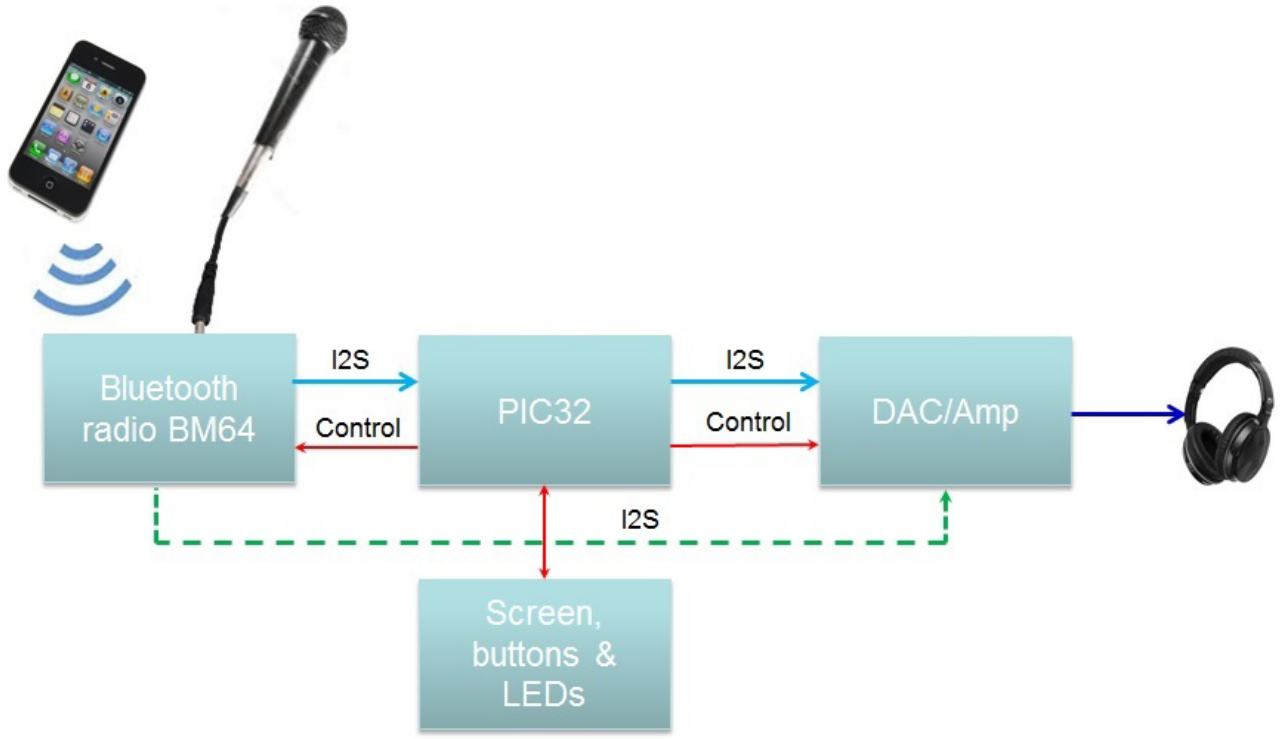
The BM64 streams I2S audio at up to 24-bit and 96 kHz, and uses a UART to receive commands from the host microcontroller (PIC32) and send events back over the same interface.

Protocols supported by the BM64 include A2DP, AVRCP, HFP, HSP, SPP, and BLE. However, this version of the driver only supports A2DP, AVRCP, HFP, and BLE.

The BM64 can be connected to a microphone (for HFP) and also has line-input; however, the latter is not supported by this driver. The multi-speaker modes of the BM64 are also not handled by this driver.

A typical interface of BM64 to a Microchip PIC32 device is provided in the following diagram:

BM64 to PIC32 Device Interface



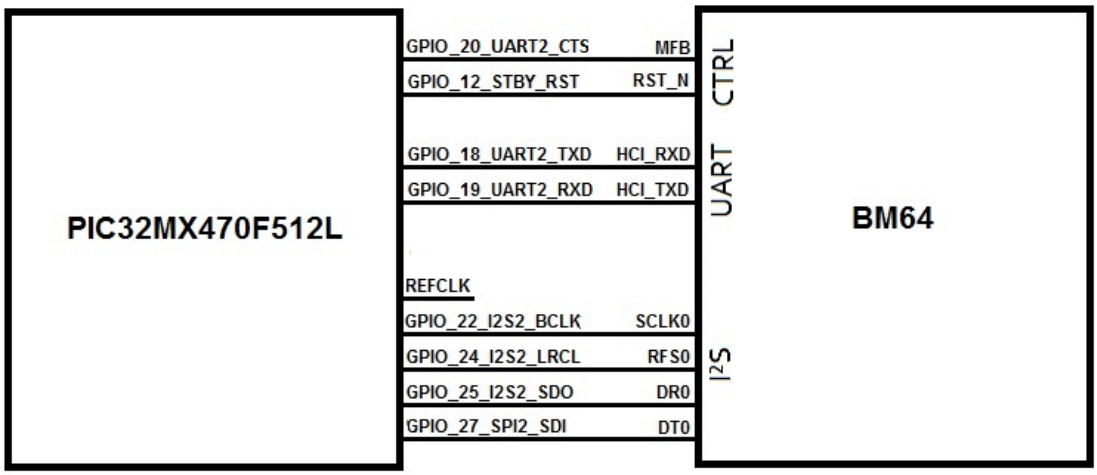
An example demonstration application using this library to interface with the BM64 for audio is BM64_a2dp_hfp, which runs on the PIC32 Bluetooth Audio Development Kit and is used to stream A2DP audio from a Bluetooth host such as a smartphone to a pair of headphones connected to the Audio DAC Daughter Board which comes with the PIC32 Bluetooth Audio Development Kit. The smartphone is controlled using the AVRCP functions of the library. That demonstration can also automatically answer a voice call coming in via Hands-Free Protocol (HFP), interrupting (and pausing) any A2DP streaming in progress.

An example demonstration application using this library to interface with the BM64 for BLE functionality is BM64_ble_comm, which is used to send a string of characters to a smartphone when one of the push buttons is pressed on the PIC32 Bluetooth Audio Development Kit, and to receive a string of characters from the smartphone and display them on the LCD of the PIC32 Bluetooth Audio Development Kit, both using the "Transparent

Service" feature of the BM64.

The following diagram shows the specific connections used in the PIC32 Bluetooth Audio Development Kit, which uses a PIC32MX470F512L microcontroller:

PIC32 Device and Module Connections



Using the Library

This topic describes the basic architecture of the BM64 Bluetooth Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_bm64.h`

The interface to the BM64 Bluetooth Driver library is defined in the `drv_bm64.h` header file. Any C language source (`.c`) file that uses the BM64 Bluetooth Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Source Files:

The BM64 Bluetooth Driver Library source files are provided in the `<install-dir>\framework\driver\bluetooth\bm64\src` directory. This folder may contain optional files and alternate implementations. Please refer to Configuring the Library for instructions on how to select optional features. and to Building the Library for instructions on how to build the library.

When the library is being used to stream A2DP audio from the BM64 to the PIC32, the BM64 must be configured as a I2S slave device. See the application BM64_bootloader demonstration application for instructions on how to do this.

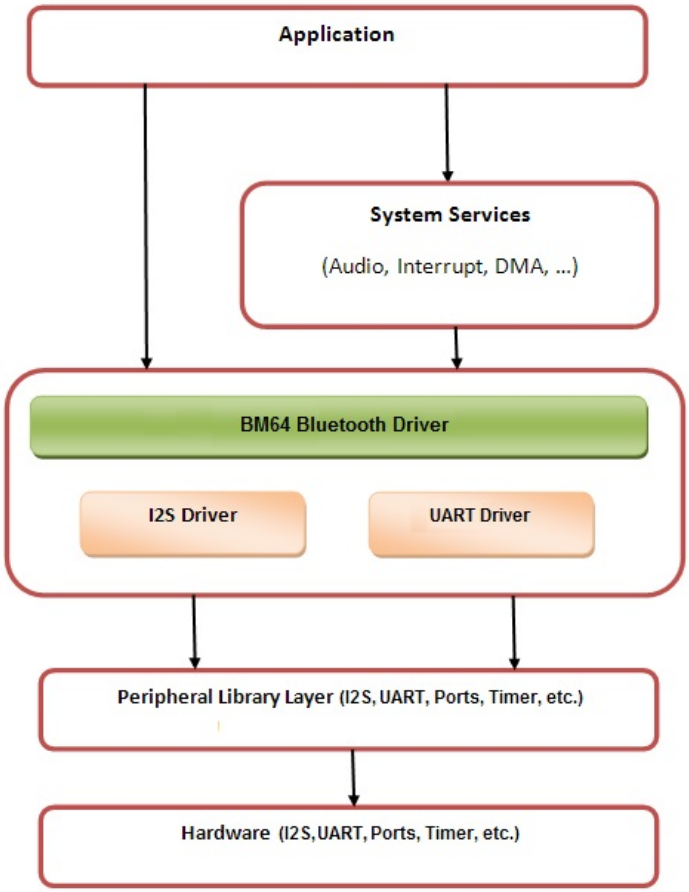
Abstraction Model

This library provides a low-level abstraction of the BM64 Bluetooth Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the BM64 Bluetooth Driver is positioned in the MPLAB Harmony framework. The BM64 Bluetooth Driver uses the USART Driver Library to control the BM64 and receive event notifications, the I2S Driver Library is used to receive audio from the BM64, and the Timer Driver for periodic timing.

BM64 Bluetooth Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The BM64 Bluetooth Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced BM64 Bluetooth module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the BM64 Bluetooth Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Data Transfer Functions	Provides data transfer functions.
Settings Functions	Provides driver specific functions for settings, such as volume control and sampling rate.
Bluetooth-specific Functions	Provides functions that are Bluetooth-specific.
AVRCP Functions	Provides functions that are used for AVRCP control.
Device Name and Address Functions	Provides functions for getting and setting the Bluetooth name and address.
BLE Functions	Provides BLE-specific functions.

How the Library Works

Provides information on how the library works.

Description

The library provides interfaces to support:

- System
- Client Setup
- Data Transfer
- Settings
- Bluetooth
- AVRCP
- Device Name and Address
- BLE

The library can be used by programs providing functionality for audio (A2DP, AVRCP and BLE), or BLE, or both.

For audio (A2DP/AVRCP/HFP), typically, there will be one simple state machine for the application and a second state machine just for the audio. After the application initializes, the audio state machine will open the BM64 Bluetooth Driver using a call to its Open function. Then, it will set up callbacks for each of two event handlers, and then open the codec driver using a call to its Open function and set up a callback from it. Then, the driver will wait until the BM64 initialization is complete, at which time the application state machine instructs the audio state machine to perform an initial buffer read from the BM64 using an AddRead call.

```
case AUDIO_STATE_OPEN:
{
    if (SYS_STATUS_READY == DRV_BT_Status())
    {
        // open BT module, including RX audio stream
        audioData.bt.handle = DRV_BT_Open(DRV_IO_INTENT_READ, DRV_BT_PROTOCOL_ALL);
        if(audioData.bt.handle != DRV_HANDLE_INVALID)
        {
            audioData.state = AUDIO_STATE_SET_BT_BUFFER_HANDLER;
        }
    }
}
break;
case AUDIO_STATE_SET_BT_BUFFER_HANDLER:
{
    DRV_BT_BufferEventHandlerSet(audioData.bt.handle, audioData.bt.bufferHandler,
                                 audioData.bt.context);
    DRV_BT_EventHandlerSet(audioData.bt.handle, audioData.bt.eventHandler, (uintptr_t)0);
                           audioData.state = AUDIO_STATE_CODEC_OPEN;
}
break;
case AUDIO_STATE_CODEC_OPEN:
{
    audioData.codec.handle =  DRV_CODEC_Open(DRV_CODEC_INDEX_0,  DRV_IO_INTENT_WRITE |
                                             DRV_IO_INTENT_EXCLUSIVE);
    if(audioData.codec.handle != DRV_HANDLE_INVALID)
    {
        audioData.state = AUDIO_STATE_CODEC_SET_BUFFER_HANDLER;
    }
}
break;
case AUDIO_STATE_CODEC_SET_BUFFER_HANDLER:
{
    _setCodecSamplingRate(DRV_BT_AUDIO_SAMPLING_RATE);
    DRV_CODEC_BufferEventHandlerSet(audioData.codec.handle, audioData.codec.bufferHandler,
                                    audioData.codec.context);
    audioData.state = AUDIO_STATE_INIT_DONE;
}
break;
case AUDIO_STATE_INIT_DONE:
{
    // waits in this state until BT initialization done and app state machine
    // calls audioStart() to set state to AUDIO_STATE_BT_SUBMIT_INITIAL_READS
    break;
}
```

After the initial buffer read has been completed, the buffer event handler for the BM64 will get a DRV_BT_BUFFER_EVENT_COMPLETE event. Once the queue has filled up, this will advance the audio state machine's state so that it adds the buffer to the codec's queue using its AddWrite function call. It then also makes a new call to the AddRead function to keep the queue filled.

When the buffer event handler for the codec gets a DRV_CODEC_BUFFER_EVENT_COMPLETE event, it will mark the buffer free for use again.

See the BM64 demonstration application, BM64_a2dp_hfp, for more information and more example code.

BLE-only applications are much simpler since they do not have to process any audio. Again typically there will be one simple state machine for the application and a second state machine just for the BLE functionality. After the application initializes, the BLE state machine will open the BM64 Bluetooth driver using a call to its Open function, then it will set up a callback for an event handler.

The application will call one of the BLE Send functions to send data to the host (smartphone). The event handler will be called whenever data has been received from the BM64, or when the connection status changes. See the BM64 demonstration application, BM64_ble_comm, for more information and example code.

System Functions

This section describes the BM64 Bluetooth driver functions for initialization, maintaining task state and returning status.

Description

Initialization

The function DRV_BM64_Initialize is called by the function SYS_Initialize, in the file system_init.c, to initialize the BM64 Bluetooth driver using constants from the generated system_config.h file.

Tasks

The function DRV_BM64_Tasks is called from the System Task Service via the function SYS_Tasks in the file system_tasks.c to maintain the driver's internal control and data interface state machine.

One can use the function DRV_BM64_TasksReq to make a power on/power off task request (DRV_BM64_REQ_SYSTEM_OFF or DRV_BM64_REQ_SYSTEM_ON).

Status

The function DRV_BM64_Status returns the BM64 Bluetooth driver status, such as SYS_STATUS_READY, SYS_STATUS_BUSY, or SYS_STATUS_ERROR. The driver should not be opened until it has been marked ready.

Example:

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
if (SYS_STATUS_READY == DRV_BT_Status())
{
// This means the driver can be opened using the
// DRV_BT_Open() function.
}
```

The BM64-specific function DRV_BM64_GetPowerStatus returns the current power status, e.g. DRV_BM64_STATUS_OFF, DRV_BM64_STATUS_OFF, and DRV_BM64_STATUS_READY. Once it returns a ready status, this means the BM64 driver has completed its internal state machine initialization and can begin processing audio.

Example:

```
case APP_STATE_WAIT_INIT:
{
// note generic version of call (DRV_BT instead of DRV_BM64) is used
if (DRV_BT_STATUS_READY == DRV_BT_GetPowerStatus())
{
appData.state=APP_STATE_IDLE;
// can start processing audio
}
}
```

Client Functions

This section describes the BM64 Bluetooth driver functions for client setup (open, close, and setting up event handlers).

Description

Open and Close

For the application to start using an instance of the module, it must call the DRV_BM64_Open function which provides a driver handle to the BM64 Bluetooth driver instance.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the BM64 Bluetooth Driver can be known by calling DRV_BM64_Status.

Example:

```
case AUDIO_STATE_OPEN:
{
```

```
    if (SYS_STATUS_READY == DRV_BT_Status())
    {
        // open BT module, including RX audio stream
        audioData.bt.handle = DRV_BT_Open(DRV_IO_INTENT_READ, DRV_BT_PROTOCOL_ALL);
        if(audioData.bt.handle != DRV_HANDLE_INVALID)
        {
            audioData.state = AUDIO_STATE_SET_BT_BUFFER_HANDLER;
        }
    }
}
```

Event Handlers

Event handlers are functions in the user's code that are used as callbacks from the driver when something occurs that the client needs to know about.

The function DRV_BM64_BufferEventHandlerSet is called by a client to identify a buffer-related event handling function for the driver to call back. The prototype for the callback is defined by DRV_BM64_BUFFER_EVENT_HANDLER. The callback will be called with the event DRV_BT_BUFFER_EVENT_COMPLETE.

The function DRV_BM64_EventHandlerSet is called by a client to identify a general event handling function for the driver to call back. The prototype for the callback is defined by DRV_BM64_EVENT_HANDLER.

For audio applications, the callback will be called with events such as DRV_BT_EVENT_VOLUME_CHANGED, DRV_BT_EVENT_SAMPLERATE_CHANGED, and DRV_BT_EVENT_PLAYBACK_STATUS_CHANGED. For BLE applications, the callback will be called for events such as DRV_BT_EVENT_BLESPP_MSG_RECEIVED and DRV_BT_EVENT_BLE_STATUS_CHANGED.

Example:

```
case APP_STATE_SET_BT_BUFFER_HANDLER:
{
    // note generic version of calls (DRV_BT instead of DRV_BM64) are used
    DRV_BT_BufferEventHandlerSet(appData.bt.handle, appData.bt.bufferHandler,
                                 appData.bt.context);
    DRV_BT_EventHandlerSet(appData.bt.handle, appData.bt.eventHandler, (uintptr_t)0);
    appData.state = APP_STATE_CODEC_OPEN;
}
```

Data Transfer Function

This section describes the BM64 Bluetooth Driver data transfer function.

Description

The function DRV_BM64_BufferAddRead schedules a non-blocking read operation. It returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully.

If the requesting client registered an event callback with the driver, the driver will issue a DRV_BM64_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_BM64_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Example:

```
case APP_STATE_BT_BUFFER_COMPLETE:
{
    if (!_bufferUsed[appData.readIndex])
    {
       //Next BT Read Queued
       // note generic version of call (DRV_BT instead of DRV_BM64) is used
       DRV_BT_BufferAddRead(appData.bt.handle, &appData.bt.readBufHandle,
               audioBuffer[appData.readIndex],   appData.bt.bufferSize);
       if(appData.bt.readBufHandle != DRV_BT_BUFFER_HANDLE_INVALID)
       {
           appData.bt.readBufHandle = DRV_BT_BUFFER_HANDLE_INVALID;
           _bufferUsed[appData.readIndex] = true;
           appData.readIndex++;
           if(appData.readIndex >= AUDIO_QUEUE_SIZE)
           {
               appData.readIndex = 0;
           }
           appData.state = APP_STATE_BT_WAIT_FOR_BUFFER_COMPLETE;
       }
    }
}
```

Settings Functions

This section describes the BM64 Bluetooth Driver functions for getting and changing settings such as volume and sample rate.

Description

The function DRV_BM64_VolumeGet returns the volume for the current mode (A2DP or HFP) in percent (0-100), and the corresponding function DRV_BM64_VolumeSet sets the volume in percent.

The functions DRV_BM64_VolumeUp and DRV_BM64_VolumeDown turn the volume up and down on the host device (e.g. smartphone) by one increment (about 3% of full-scale). Either of these will result in a callback with the event DRV_BM64_EVENT_VOLUME_CHANGED specifying the new volume setting.

Example:

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    // bump the volume up one notch based on a button press
    if (BSP_SwitchStateGet(BSP_SWITCH_2)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_volumeUp(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}

. . .

// later, a call will come back to the event handler callback function
// (previously set up via a call to DRV_BM64_EventHandlerSet)
static void _BLEEventHandler(DRV_BT_EVENT event, uint32_t param, uintptr_t context)
{
    switch(event)
    {
        case DRV_BM64_EVENT_VOLUME_CHANGED:
        {
            uint16_t volume7bits = (127*param)/100;     // convert to 7 bits
            DRV_AK4384_VolumeSet(audioData.codec.handle,         // update codec's volume
                    DRV_AK4384_CHANNEL_LEFT_RIGHT,volume7bits);
            laString tempStr;
            char buf[5];
            sprintf(buf,"%3d%%",param);
            laWidget_SetVisible((laWidget*)GFX_VOLUME_VALUE, LA_TRUE);
            tempStr = laString_CreateFromCharBuffer(buf, &LiberationSans12);
            laLabelWidget_SetText(GFX_VOLUME_VALUE, tempStr);    // update screen
            laString_Destroy(&tempStr);
        }
   }
}
```

Sample Rate

This section describes the functions for getting and setting the sampling rate (e.g., 8000, 44100, or 48000 Hz) as a 32-bit integer.

Description

The function DRV_BM64_EnterBTPairingMode is used to enter into pairing mode. Once the BM64 is paired with a device, it will automatically attempt to connect with it again on the next power cycle.

Calling DRV_BM64_DisconnectAllLinks will disconnect the BM64 from the host (smartphone) but will not erase the pairing. So another call to the function DRV_BM64_LinkLastDevice will reconnect. However calling the function DRV_BM64_ForgetAllLinks will erase all pairing information, and another call to DRV_BM64_EnterBTPairingMode will be required to re-establish a connection.

Example:

```
    case BUTTON_STATE_PRESSED:      // (debouncing not shown)
    {
        // initiate pairing with a button press
        if (BSP_SwitchStateGet(BSP_SWITCH_1)==BSP_SWITCH_STATE_PRESSED))
        {
            // note generic version of call (DRV_BT instead of DRV_BM64) is used
            DRV_BT_EnterBTPairingMode(appData.bt.handle);
            appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
```

```
        }
    }
```

The function DRV_BM64_GetLinkStatus returns the current link status, returning an 8-bit value containing the current link status defined by DRV_BM64_LINKSTATUS enum. This can be used to restrict calls to AVRCP functions only when an AVRCP link is established.

Example:

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
if (DRV_BT_GetLinkStatus(appData.bt.handle) & DRV_BT_AVRCP_LINK_STATUS)
{
    DRV_BT_CancelForwardOrRewind(appData.bt.handle);
}
```

AVRCP Functions

This section describes the functions for getting and setting the Bluetooth device's name and address.

Description

The function DRV_BM64_SetBDName is called to set a temporary Bluetooth device name from an ASCII string buffer. The function DRV_BM64_GetBDName is called to get the current Bluetooth device name, and DRV_BM64_GetBDAddress is called to get the Bluetooth device address.

Example:

```
laString tempStr;
char buf [DRV_BT_MAXBDNAMESIZE+1];
  // note generic version of calls (DRV_BT instead of DRV_BM64) are used
DRV_BT_GetBDName(appData.bt.handle, buf, DRV_BT_MAXBDNAMESIZE+1);
tempStr = laString_CreateFromCharBuffer(buf, &LiberationSans12);
laLabelWidget_SetText(GFX_BTNAME_VALUE, tempStr);   // display BT name
laString_Destroy(&tempStr);
DRV_BT_GetBDAddress(appData.bt.handle, buf);
tempStr = laString_CreateFromCharBuffer(buf, &LiberationSans12);
laLabelWidget_SetText(GFX_BTADDRESS_VALUE, tempStr);    // display BT address
laString_Destroy(&tempStr);
```

BLE Functions

This section describes the functions specific to Bluetooth Low Energy (BLE) operations, such as sending and receiving data, and BLE connection-related operations.

Description

The function DRV_BM64_ReadByteFromBLE is used to receive data one byte at a time; the function DRV_BM64_ReadDataFromBLE is used to receive multiple bytes. Each of them return a Boolean, which is true if data is returned or false if there is no data to return. You can use the function DRV_BM64_ClearBLEData to clear out the receive buffer before starting.

Example:

```
uint8_t byte;
// note generic versions of calls (DRV_BT instead of DRV_BM64) are used
DRV_BT_ClearBLEData(appData.bt.handle);
// wait for byte to arrive
while (!DRV_BT_ReadByteFromBLE(appData.bt.handle, &byte))
{
    // should have some sort of way to break out of here if byte never arrives
}
```

Sending Data

The function DRV_BM64_SendByteOverBLE Is used to send one byte of data at a time; the function DRV_BM64_SendDataOverBLE is used to send multiple bytes of data.

Example:

```
#define BUFSIZE    100
uint8_t buf [BUFSIZE];
 // (code goes here to fill in buffer with data)

// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_SendDataOverBLE(appData.bt.handle, buf, BUFSIZE);
```

Connection Status

The function DRV_BM64_BLE_EnableAdvertising is called to enable or disable BLE advertising.

The function DRV_BM64_BLE_QueryStatus queries the BM64 to respond with a DRV_BM64_EVENT_BLE_STATUS_CHANGED event, which will indicate if the BM64 BLE status is standby, advertising, scanning or connected.

Example:

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_BLE_QueryStatus(appData.bt.handle);
. . .
// later, a call will come back to the event handler callback function
// (previously set up via a call to DRV_BM64_EventHandlerSet)
static void _BLEEventHandler(DRV_BT_EVENT event, uint32_t param, uintptr_t context)
{
    switch(event)
    {
        case DRV_BT_EVENT_BLE_STATUS_CHANGED:
        {
            // do case switch based on param variable
        switch(param)
        {
            case DRV_BM64_BLE_STATUS_STANDBY:
            case DRV_BM64_BLE_STATUS_SCANNING:
                laWidget_SetVisible((laWidget*)GFX_CONNECTED, LA_FALSE);
                laWidget_SetVisible((laWidget*)GFX_PAIRED, LA_FALSE);
                laWidget_SetVisible((laWidget*)GFX_NOPAIR_NOCONNECTION, LA_TRUE);
                break;
            case DRV_BM64_BLE_STATUS_ADVERTISING:
                laWidget_SetVisible((laWidget*)GFX_CONNECTED, LA_FALSE);
                laWidget_SetVisible((laWidget*)GFX_PAIRED, LA_TRUE);    // actually, advertising
                laWidget_SetVisible((laWidget*)GFX_NOPAIR_NOCONNECTION, LA_FALSE);
                break;
            case DRV_BM64_BLE_STATUS_CONNECTED:
                laWidget_SetVisible((laWidget*)GFX_CONNECTED, LA_TRUE);
                laWidget_SetVisible((laWidget*)GFX_PAIRED, LA_FALSE);
                laWidget_SetVisible((laWidget*)GFX_NOPAIR_NOCONNECTION, LA_FALSE);
                break;
        }
   }
}
```

Configuring the Library

Macros

	Name	Description
	INCLUDE_BM64_BLE	Identifies whether the driver should include BLE
	INCLUDE_BM64_I2S	Identifies whether the driver should include HFP,A2DP,AVRCP functionality.
	INCLUDE_DEPRECATED_MMI_COMMANDS	Identifies whether the driver should use deprecated MMI commands.

Description

The configuration of the BM64 Bluetooth Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the BM64 Bluetooth Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the BM64 Bluetooth Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

INCLUDE_BM64_BLE Macro

Identifies whether the driver should include BLE

File

drv_bm64_config_template.h

C

```
#define INCLUDE_BM64_BLE
```

Description

Include BLE features?

Identifies whether the driver should include BLE (Bluetooth Low Energy) functions.

This option currently does not have any effect on the code size.

true (checked, default) - include BLE functionality. false (unchecked) - do not include BLE functionality.

Remarks

None

INCLUDE_BM64_I2S Macro

Identifies whether the driver should include HFP,A2DP,AVRCP functionality.

File

drv_bm64_config_template.h

C

```
#define INCLUDE_BM64_I2S
```

Description

Include HFP,A2DP,AVRCP protocols?

Identifies whether the driver should include the interface to support HFP, A2DP and AVRCP protocols, which by default also brings in the I2S driver and the default codec based on the BSP selected.

If you are building a BLE-only application, uncheck this option.

true (checked, default) - include HFP,A2DP,AVRCP functionality. false (unchecked) - do not include HFP,A2DP,AVRCP functionality.

Remarks

None

INCLUDE_DEPRECATED_MMI_COMMANDS Macro

Identifies whether the driver should use deprecated MMI commands.

File

drv_bm64_config_template.h

C

```
#define INCLUDE_DEPRECATED_MMI_COMMANDS
```

Description

Use Deprecated MMI Commands?

There are currently two versions of the BM64 Audio UART Command Set, which is used by the PIC32 to send commands to the BM64 module and receive responses (events) back from the BM64. The original is version 1.00 and the updated one is version 2.0x. Version 2.0x deprecates some MMI commands, and adds some new commands to replace them.

If the DRV_BM64_PlayPreviousSong and DRV_BM64_PlayNextSong functions are not working but other AVRCP functions are working properly, try unchcing this option.

true (checked, default) - use deprecated MMI commands. false (unchecked) - do not deprecated MMI commands.

Remarks

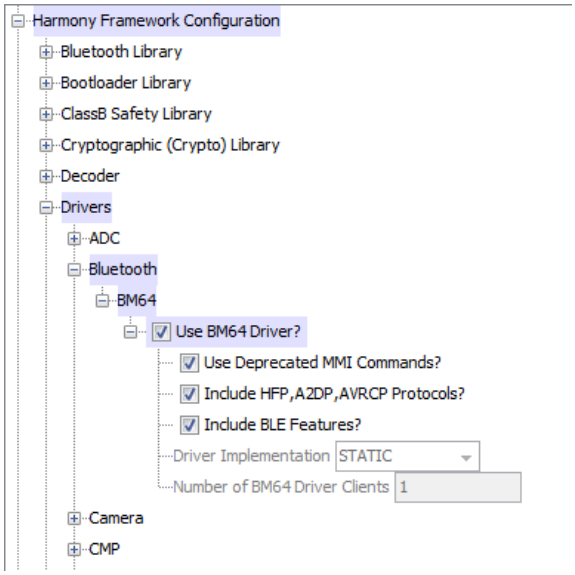
None

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following figure shows an example MHC configuration for the BM64 Bluetooth Driver.



The option **Include HFP,A2DP,AVRCP protocols?** identifies whether the driver should include the interface to support HFP, A2DP and AVRCP protocols, which by default also brings in the I2S driver and the default codec based on the BSP selected. If you are building a BLE-only application, uncheck this option.

The option **Include BLE features?** identifies whether the driver should include BLE functions. If you are not using any BLE functionality, uncheck this option.

When **Use BM64 Driver?** is selected, and you have already selected the PIC32 Bluetooth Audio Development Kit (AK4384), the proper configuration for the AK4384, I2S, and Timer will have already been made for you, including:

- Under Drivers/CODEC,
- Use_Codec_AK4384 selected
- I2S Driver (used for data interface interface) instance set to DRV_I2S_INDEX_1
- Under I2S,
- Use I2S Driver Selected
- DMA Mode Selected
- Transmit DMA Support Selected
- Receive DMA Support Selected
- Enable DMA Channel Interrupts selected
- Sampling Rate set to 8000
- Number of I2S Instances set to 2
- I2S Driver Instance 0 selected
- I2S Module ID set to SPI_ID_2 (BM64 Module as wired on BTADK)
- Audio Protocol Mode set to DRV_I2S_AUDIO_I2S
- I2S Driver Instance 1 selected
- I2S Module ID set to SPI_ID_1 (AK4384 DAC Module as wired on BTADK)
- Audio Protocol Mode set to DRV_I2S_AUDIO_LFET_JUSTIFIED

Building the Library

This section lists the files that are available in the BM64 Bluetooth Driver Library.

Description

This section lists the files that are available in the `/src` folder of the BM64 Bluetooth Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/bluetooth/bm64`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_bm64.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
drv_bm64_ble.h	Header file for the internal functions of the driver related to BLE.
drv_bm64_command_decode.h	Header file for the internal functions of the driver for decoding events from the BM64.
drv_bm64_command_send.h	Header file for the internal functions of the driver for sending commands to the BM64
drv_bm64_gpio.h	Header file for the internal functions of the driver related to the BM64's control pins.
drv_bm64_line_in.h	Header file for the internal functions of the driver related to the BM64's line in input.
./src/framework/driver/bluetooth/bm64/drv_bm64_local.h	Header file for the functions local to the BM64 driver (generated from template).
drv_bm64_sha1.h	Header file for the internal functions of the driver for performing SHA hashes.
drv_bm64_uart.h	Header file for the internal functions of the driver related to the BM64's UART interface.
./src/framework/driver/bluetooth/bm64/src/drv_bm64.c	Main source implementation file for the driver (generated from template).
src/drv_bm64_ble.c	Source file for the internal functions of the driver related to BLE.
src/drv_bm64_command_decode.c	Source file for the internal functions of the driver for decoding events from the BM64.
src/drv_bm64_command_send.c	Source file for the internal functions of the driver for sending commands to the BM64
src/drv_bm64_gpio.c	Source file for the internal functions of the driver related to the BM64's control pins.
src/drv_bm64_line_in.c	Source file for the internal functions of the driver related to the BM64's line in input.
src/drv_bm64_sha1.c	Source file for the internal functions of the driver for performing SHA hashs.
src/drv_bm64_uart.c	Source file for the internal functions of the driver related to the BM64's UART interface.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
There are no optional files for this driver.	N/A

Module Dependencies

The BM64 Bluetooth Driver Library depends on the following modules:

- I2S Driver Library
- Timer Driver Library
- USART Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_BM64_GetPowerStatus	Gets the current status of the BM64 Bluetooth driver module (BM64-specific).
	DRV_BM64_Initialize	Initializes hardware and data for the instance of the BM64 Bluetooth module
	DRV_BM64_Status	Gets the current system status of the BM64 Bluetooth driver module.
	DRV_BM64_TaskReq	Make a power on/power off task request.
	DRV_BM64_Tasks	Maintains the driver's control and data interface state machine.

b) Client Setup Functions

	Name	Description
	DRV_BM64_BufferEventHandlerSet	This function allows a client to identify a event handling function for the driver to call back.
	DRV_BM64_Close	Close an opened-instance of the BM64 Bluetooth driver.
	DRV_BM64_EventHandlerSet	This function allows a client to identify an event handling function for the driver to call back.
	DRV_BM64_Open	Open the specified BM64 driver instance and returns a handle to it

c) Data Transfer Functions

	Name	Description
	DRV_BM64_BufferAddRead	Schedule a non-blocking driver read operation.

d) Settings Functions

	Name	Description
	DRV_BM64_SamplingRateGet	Return the current sampling rate.
	DRV_BM64_SamplingRateSet	Set the current sampling rate.
	DRV_BM64_volumeDown	Turn the volume down on the host device.
	DRV_BM64_VolumeGet	returns 7-bit value 0-127
	DRV_BM64_VolumeSet	Set current volume.
	DRV_BM64_volumeUp	Turn the volume up on the host device.

e) Bluetooth-specific Functions

	Name	Description
	DRV_BM64_DisconnectAllLinks	Disconnect all links.
	DRV_BM64_EnterBTPairingMode	Enter Bluetooth pairing mode.
	DRV_BM64_ForgetAllLinks	Forget all pairings.
	DRV_BM64_GetLinkStatus	Return link status.
	DRV_BM64_LinkLastDevice	Link last device.

f) AVRCP Functions

	Name	Description
	DRV_BM64_CancelForwardOrRewind	Cancel previous fast forward or rewind request.
	DRV_BM64_FastForward	Fast forward the media.
	DRV_BM64_GetPlayingStatus	Return the current playing status of the device.
	DRV_BM64_Pause	Pause playback.
	DRV_BM64_Play	Start playback.
	DRV_BM64_PlayNextSong	Play the next song.
	DRV_BM64_PlayPause	Toggle play/pause mode.
	DRV_BM64_PlayPreviousSong	Play the previous song.
	DRV_BM64_Rewind	Rewind the media.
	DRV_BM64_Stop	Stop playback.

g) Device Name and Address Functions

	Name	Description
	DRV_BM64_GetBDAddress	Return the Bluetooth address.
	DRV_BM64_GetBDName	Return Bluetooth device name.
	DRV_BM64_SetBDName	Set the Bluetooth device name.

h) BLE Functions

	Name	Description
	DRV_BM64_ClearBLEData	Clear the BLE receive buffer.
	DRV_BM64_ReadByteFromBLE	Read a byte over BLE.
	DRV_BM64_ReadDataFromBLE	Read data over BLE.
	DRV_BM64_SendByteOverBLE	Send a byte over BLE.
	DRV_BM64_SendDataOverBLE	Send data over BLE.
	DRV_BM64_BLE_QueryStatus	Query BM64 LE status.
	DRV_BM64_BLE_EnableAdvertising	Enable or disable advertising.

i) Data Types and Constants

	Name	Description
	DRV_BM64_BUFFER_EVENT	This is macro DRV_BM64_BUFFER_EVENT.
	DRV_BM64_BUFFER_EVENT_COMPLETE	This is macro DRV_BM64_BUFFER_EVENT_COMPLETE.
	DRV_BM64_BUFFER_HANDLE	This is macro DRV_BM64_BUFFER_HANDLE.

	DRV_BM64_BUFFER_HANDLE_INVALID	This is macro DRV_BM64_BUFFER_HANDLE_INVALID.
	DRV_BM64_DATA32	BM64 defines based on I2S interface
	DRV_BM64_MAXBDNAMESIZE	
	DRV_BM64_BUFFER_EVENT_HANDLER	prototype for callback for DRV_BM64_BufferEventHandlerSet
	DRV_BM64_DRVR_STATUS	BM64 driver status
	DRV_BM64_EVENT	events that can be returned to a client via callback
	DRV_BM64_EVENT_HANDLER	prototype for callback for DRV_BM64_EventHandlerSet
	DRV_BM64_LINKSTATUS	BM64 link status
	DRV_BM64_PLAYINGSTATUS	This is type DRV_BM64_PLAYINGSTATUS.
	DRV_BM64_PROTOCOL	BM64 protocols
	DRV_BM64_REQUEST	BM64 power on/off request
	DRV_BM64_SAMPLE_FREQUENCY	BM64 sample frequency
	DRV_BM64_BLE_STATUS	This is type DRV_BM64_BLE_STATUS.

Description

This section describes the API functions of the BM64 Bluetooth Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_BM64_GetPowerStatus Function

Gets the current status of the BM64 Bluetooth driver module (BM64-specific).

File

drv_bm64.h

C

```
DRV_BM64_DRVR_STATUS DRV_BM64_GetPowerStatus();
```

Returns

Driver status, encoded as type DRV_BM64_DRVR_STATUS enum.

Description

Function DRV_BM64_GetPowerStatus:

DRV_BM64_DRVR_STATUS DRV_BM64_GetPowerStatus(void);

This routine provides the current status (power on/off/ready) of the BM64 Bluetooth driver module passed back as type DRV_BM64_DRVR_STATUS enum.

Remarks

A status of DRV_BT_STATUS_READY means the drivers state machine has finished initialization and is ready to stream audio.

Preconditions

DRV_BM64_Initialize must have been called to initialize the driver instance.

Example

```
case APP_STATE_WAIT_INIT:
{
   // note generic version of call (DRV_BT instead of DRV_BM64) is used
   if (DRV_BT_STATUS_READY == DRV_BT_GetPowerStatus())
   {
       appData.state=APP_STATE_IDLE;
       // start can processing audio
   }
}
break;
```

DRV_BM64_Initialize Function

Initializes hardware and data for the instance of the BM64 Bluetooth module

File

drv_bm64.h

C

```
void DRV_BM64_Initialize();
```

Returns

None.

Description

Function DRV_BM64_Initialize:

void DRV_BM64_Initialize(void);

This routine initializes the BM64 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other BM64 driver routine is called. This routine should only be called once during system initialization. This routine will never block for hardware access.

Preconditions

None.

Example

```
// (in SYS_Initialize, system_init.c)*

DRV_BM64_Initialize();
```

DRV_BM64_Status Function

Gets the current system status of the BM64 Bluetooth driver module.

File

drv_bm64.h

C

```
SYS_STATUS DRV_BM64_Status();
```

Returns

Driver status, encoded as type SYS_STATUS enum:

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed SYS_STATUS_ERROR - Indicates that the specified module is in an error state *

Description

Function DRV_BM64_Status:

SYS_STATUS DRV_BM64_Status(void);

This routine provides the current status of the BM64 Bluetooth driver module, passed back as type SYS_STATUS.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

None.

Example

```
* // note generic version of call (DRV_BT instead of DRV_BM64) is used
```

```
if (SYS_STATUS_READY == DRV_BT_Status())
{
    // This means the driver can be opened using the
    // DRV_BT_Open() function.
}
```

DRV_BM64_TaskReq Function

Make a power on/power off task request.

File

drv_bm64.h

C

```
void DRV_BM64_TaskReq(DRV_BM64_REQUEST request);
```

Returns

None.

Description

Function DRV_BM64_TaskReq:

void DRV_BM64_TaskReq(DRV_BM64_REQUEST request);

Make a power on/power off task request using the DRV_BM64_REQUEST enum.

Remarks

None.

Preconditions

DRV_BM64_Initialize must have been called to initialize the driver instance.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_TaskReq(DRV_BM64_REQ_SYSTEM_ON);
```

Parameters

Parameters	Description
request	power on/off request of type DRV_BM64_REQUEST

DRV_BM64_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_bm64.h

C

```
void DRV_BM64_Tasks();
```

Returns

None.

Description

Function DRV_BM64_Tasks:

void DRV_BM64_Tasks(void);

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations.

This function should be called from the SYS_Tasks() function.

Remarks

This routine is not normally called directly by an application. Instead it is called by the system's Tasks routine (SYS_Tasks).

Preconditions

None.

Example

```
// (in SYS_Tasks, system_tasks.c)

// Maintain Device Drivers
DRV_BM64_Tasks();
```

b) Client Setup Functions

DRV_BM64_BufferEventHandlerSet Function

This function allows a client to identify a event handling function for the driver to call back.

File

drv_bm64.h

C

```
void DRV_BM64_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_BM64_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle);
```

Returns

None.

Description

Function DRV_BM64_BufferEventHandlerSet:

void DRV_BM64_EventHandlerSet(DRV_HANDLE handle, const DRV_BM64_BUFFER_EVENT_HANDLER eventHandler, const uintptr_t contextHandle);

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_BM64_BufferAddRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The context parameter contains a handle to the client context, provided at the time the event handling function is registered using the DRV_BM64_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client.

The event handler should be set before the client performs any "BM64 Bluetooth Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case APP_STATE_SET_BT_BUFFER_HANDLER:
{
    // note generic version of call (DRV_BT instead of DRV_BM64) is used
    DRV_BT_BufferEventHandlerSet(appData.bt.handle,
                                 appData.bt.bufferHandler,
                                 appData.bt.context);

    DRV_BT_EventHandlerSet(appData.bt.handle,
                                 appData.bt.eventHandler,
                                 (uintptr_t)0);

    appData.state = APP_STATE_CODEC_OPEN;
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
eventHandler	pointer to a function to be called back (prototype defined by DRV_BM64_BUFFER_EVENT_HANDLER)
contextHandle	handle to the client context

DRV_BM64_Close Function

Close an opened-instance of the BM64 Bluetooth driver.

File

drv_bm64.h

C

```
void DRV_BM64_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_Close:

void DRV_BM64_Close(DRV_HANDLE handle);

This routine closes an opened-instance of the BM64 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_BM64_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_Close(appData.bt.handle);
*
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_EventHandlerSet Function

This function allows a client to identify an event handling function for the driver to call back.

File

drv_bm64.h

C

```
void DRV_BM64_EventHandlerSet(DRV_HANDLE handle, const DRV_BM64_EVENT_HANDLER eventHandler, const uintptr_t
contextHandle);
```

Returns

None.

Description

Function DRV_BM64_EventHandlerSet:

void DRV_BM64_EventHandlerSet(DRV_HANDLE handle, const DRV_BM64_EVENT_HANDLER eventHandler, const uintptr_t contextHandle);

This function allows a client to identify a command event handling function for the driver to call back when an event has been received from the BM64.

The context parameter contains a handle to the client context, provided at the time the event handling function is registered using the DRV_BM64_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client.*

The event handler should be set before the client performs any "BM64 Bluetooth Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when an event has occurred, it does not need to register a callback.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case APP_STATE_SET_BT_BUFFER_HANDLER:
{
    DRV_BT_BufferEventHandlerSet(appData.bt.handle,
                                 appData.bt.bufferHandler,
                                 appData.bt.context);

    // note generic version of call (DRV_BT instead of DRV_BM64) is used
    DRV_BT_EventHandlerSet(appData.bt.handle,
                                 appData.bt.eventHandler,
                                 (uintptr_t)0);

    appData.state = APP_STATE_CODEC_OPEN;
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
eventHandler	pointer to a function to be called back (prototype defined by DRV_BM64_EVENT_HANDLER)
contextHandle	handle to the client context

DRV_BM64_Open Function

Open the specified BM64 driver instance and returns a handle to it

File

drv_bm64.h

C

```
DRV_HANDLE DRV_BM64_Open(const DRV_IO_INTENT ioIntent, const DRV_BM64_PROTOCOL protocol);
```

Returns

valid handle to an opened BM64 device driver unique to client

Description

Function DRV_BM64_Open:

DRV_HANDLE DRV_BM64_Open(const DRV_IO_INTENT ioIntent, const DRV_BM64_PROTOCOL protocol);

This routine opens the specified BM64 Bluetooth driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

Only DRV_IO_INTENT_READ is a valid ioIntent option as the BM64 Bluetooth driver audio stream is read-only.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_BM64_Close routine is called. This routine will never block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Currently only one client is allowed at a time.

Preconditions

DRV_BM64_Initialize must have been called to initialize the driver instance.

Example

```
case APP_STATE_OPEN:
{
    if (SYS_STATUS_READY == DRV_BT_Status())
    {
        // open BT module, including RX audio stream
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        appData.bt.handle = DRV_BT_Open(DRV_IO_INTENT_READ, DRV_BT_PROTOCOL_ALL);

        if(appData.bt.handle != DRV_HANDLE_INVALID)
        {
            appData.state = APP_STATE_SET_BT_BUFFER_HANDLER;
        }
        else
        {
            // Got an Invalid Handle.  Wait for BT module to Initialize
        }
    }
}
break;
```

Parameters

Parameters	Description
ioIntent	valid handle to an opened BM64 device driver unique to client
protocol	specifies which protocol(s) the client intends to use with this driver. One of the various DRV_BM64_PROTOCOL enum values, including DRV_BM64_PROTOCOL_ALL.

c) Data Transfer Functions

DRV_BM64_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_bm64.h

C

```
void DRV_BM64_BufferAddRead(const DRV_HANDLE handle, DRV_BM64_BUFFER_HANDLE * bufferHandle, void * buffer,
size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_BM64_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_BM64_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_BM64_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_BM64_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the BM64 Driver Buffer Event Handler that is registered by this

client. It should not be called in the event handler associated with another BM64 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case APP_STATE_BT_BUFFER_COMPLETE:
{
   //BT RX
   if (!_bufferUsed[appData.readIndex])
   {
       //Next BT Read Queued
       // note generic version of call (DRV_BT instead of DRV_BM64) is used
       DRV_BT_BufferAddRead(appData.bt.handle,
                            &appData.bt.readBufHandle,
                            audioBuffer[appData.readIndex],
                            appData.bt.bufferSize);

       if(appData.bt.readBufHandle != DRV_BT_BUFFER_HANDLE_INVALID)
       {
           appData.bt.readBufHandle = DRV_BT_BUFFER_HANDLE_INVALID;
           _bufferUsed[appData.readIndex] = true;

           //QUEUE HEAD Index (for next BT read)
           appData.readIndex++;
           if(appData.readIndex >= AUDIO_QUEUE_SIZE)
           {
               appData.readIndex = 0;
           }
           appData.state = APP_STATE_BT_WAIT_FOR_BUFFER_COMPLETE;
       }
       else
       {
           SYS_DEBUG(0, "BT Buffer Read FAILED!!!");
       }
   }
   else
   {
       //Overrun -- Wait for Read buffer to become available.
       SYS_DEBUG(0, "Buffer Overrunrn");
   }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
bufferHandle	pointer to an argument that contains the return buffer handle
buffer	pointer to buffer that will contain received data
size	buffer size in bytes.

Function

void DRV_BM64_BufferAddRead(const DRV_HANDLE handle,
DRV_BM6_BUFFER_HANDLE *bufferHandle, void *buffer, size_t size)

d) Settings Functions

DRV_BM64_SamplingRateGet Function

Return the current sampling rate.

File

drv_bm64.h

C

```
uint32_t DRV_BM64_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_SamplingRateGet:

uint32_t DRV_BM64_SamplingRateGet(DRV_HANDLE handle);

Return the current sampling rate as a 32-bit integer.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t sampleRate;

// note generic version of call (DRV_BT instead of DRV_BM64) is used
sampleRate = DRV_BT_SamplingRateGet(appData.bt.handle);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_SamplingRateSet Function

Set the current sampling rate.

File

drv_bm64.h

C

```
void DRV_BM64_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

Function DRV_BM64_SamplingRateSet:

void DRV_BM64_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);

Set the current sampling rate (passed as a 32-bit integer).

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// set sample rate to 44.1 kHz
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_SamplingRateSet(appData.bt.handle, 44100);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
samplingRate	sampling rate in Hz (8000, 16000, 44100 or 48000)

DRV_BM64_volumeDown Function

Turn the volume down on the host device.

File

drv_bm64.h

C

```
void DRV_BM64_volumeDown(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_VolumeDown:

void DRV_BM64_VolumeDown(const DRV_HANDLE handle);

Turn the volume down on the host device by one increment (about 3% of full-scale).

Remarks

This will result in a callback with the event DRV_BM64_EVENT_VOLUME_CHANGED specifying the new volume setting for the codec.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_2)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_volumeUp(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_VolumeGet Function

File

drv_bm64.h

C

```
uint8_t DRV_BM64_VolumeGet(const DRV_HANDLE handle);
```

Description

returns 7-bit value 0-127

DRV_BM64_VolumeSet Function

Set current volume.

File

drv_bm64.h

C

```
void DRV_BM64_VolumeSet(const DRV_HANDLE handle, uint8_t volume);
```

Returns

None.

Description

Function DRV_BM64_VolumeSet:

void DRV_BM64_VolumeSet(const DRV_HANDLE handle, uint8_t volume);

Set volume for current mode (A2DP, HFP etc.) in percent (0-100).

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used *
volume = DRV_BT_VolumeGet(appData.bt.handle,50);       // set volume to 50%
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
volume	volume level in percent, 0-100

DRV_BM64_volumeUp Function

Turn the volume up on the host device.

File

drv_bm64.h

C

```
void DRV_BM64_volumeUp(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_VolumeUp:

void DRV_BM64_VolumeUp(const DRV_HANDLE handle);

Turn the volume up on the host device by one increment (about 3% of full-scale).

Remarks

This will result in a callback with the event DRV_BM64_EVENT_VOLUME_CHANGED specifying the new volume setting for the codec.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_1)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_volumeUp(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

e) Bluetooth-specific Functions

DRV_BM64_DisconnectAllLinks Function

Disconnect all links.

File

drv_bm64.h

C

```
void DRV_BM64_DisconnectAllLinks(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_DisconnectAllLinks:

void DRV_BM64_DisconnectAllLinks(const DRV_HANDLE handle);

Disconnect all current links to a Bluetooth host.

Remarks

Does not unpair the device, just disconnects. Use DRV_BM64_LinkLastDevice to reconnect. Use DRV_BM64_ForgetAllLinks to forget all pairings.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_2)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_DisconnectAllLinks(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_EnterBTPairingMode Function

Enter Bluetooth pairing mode.

File

drv_bm64.h

C

```
void DRV_BM64_EnterBTPairingMode(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_EnterBTPairingMode:

void DRV_BM64_EnterBTPairingMode(const DRV_HANDLE handle);

Starting the pairing process, making this BM64 available for pairing with a Bluetooth host.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_1)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_EnterBTPairingMode(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_ForgetAllLinks Function

Forget all pairings.

File

drv_bm64.h

C

```
void DRV_BM64_ForgetAllLinks(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_ForgetAllLinks:

void DRV_BM64_ForgetAllLinks(const DRV_HANDLE handle);

Forget (erase) all links and pairings stored in EEPROM.

Remarks

After this is called, one must call DRV_BM64_EnterBTPairingMode to establish a connection to a Bluetooth host again.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_2)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_ForgetAllLinks(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_GetLinkStatus Function

Return link status.

File

drv_bm64.h

C

```
DRV_BM64_LINKSTATUS DRV_BM64_GetLinkStatus(const DRV_HANDLE handle);
```

Returns

8-bit value defined by DRV_BM64_LINKSTATUS enum.

Description

Function DRV_BM64_GetLinkStatus:

DRV_BM64_LINKSTATUS DRV_BM64_GetLinkStatus(const DRV_HANDLE handle);

Returns a 8-bit value containing current link status as bit flags for SCO (bit 0), ACL, HFP, A2DP, AVRCP, SPP, IAP, MAP (bit 7)

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        DRV_BT_PLAYINGSTATUS playingStatus = DRV_BT_GetPlayingStatus(appData.bt.handle);
        if ((playingStatus==DRV_BT_PLAYING_FF)||(playingStatus==DRV_BT_PLAYING_FR))
        {
           // note generic version of call (DRV_BT instead of DRV_BM64) is used
           if (DRV_BT_GetLinkStatus(appData.bt.handle) & DRV_BT_AVRCP_LINK_STATUS)
           {
               DRV_BT_CancelForwardOrRewind(appData.bt.handle);
           }
        }
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_LinkLastDevice Function

Link last device.

File

drv_bm64.h

C

```
void DRV_BM64_LinkLastDevice(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_LinkLastDevice:

void DRV_BM64_LinkLastDevice(const DRV_HANDLE handle);

Link (connect) to last device that was previously linked.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_2)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_LinkLastDevice(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

f) AVRCP Functions

DRV_BM64_CancelForwardOrRewind Function

Cancel previous fast forward or rewind request.

File

drv_bm64.h

C

```
void DRV_BM64_CancelForwardOrRewind(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_CancelForwardOrRewind:

void DRV_BM64_CancelForwardOrRewind(const DRV_HANDLE handle);

Send an AVRCP command to the host device to cancel a previous fast forward or rewind request.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
```

```
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_CancelForwardOrRewind(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_FastForward Function

Fast forward the media.

File

drv_bm64.h

C

```
void DRV_BM64_FastForward(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_FastForward:

void DRV_BM64_FastForward(const DRV_HANDLE handle);

Send an AVRCP command to the host device to Fast forward the media.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_5)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_FastForward(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_GetPlayingStatus Function

Return the current playing status of the device.

File

drv_bm64.h

C

```
DRV_BM64_PLAYINGSTATUS DRV_BM64_GetPlayingStatus(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_GetPlayingStatus:

void DRV_BM64_GetPlayingStatus(const DRV_HANDLE handle);

Return the current AVRCP playing status of the device, e.g. stopped, playing, paused, fast forward or rewind, encoded as as the enum DRV_BM64_PLAYINGSTATUS.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_PLAYINGSTATUS playingStatus = DRV_BT_GetPlayingStatus(appData.bt.handle);
        if ((playingStatus==DRV_BT_PLAYING_FF)||(playingStatus==DRV_BT_PLAYING_FR))
        {
           if (DRV_BT_GetLinkStatus(appData.bt.handle) & DRV_BT_AVRCP_LINK_STATUS)
           {
               DRV_BT_CancelForwardOrRewind(appData.bt.handle);
           }
        }
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_Pause Function

Pause playback.

File

drv_bm64.h

C

```
void DRV_BM64_Pause(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_Pause:

void DRV_BM64_Pause(const DRV_HANDLE handle);

Send an AVRCP command to the host device to pause.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_Pause(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_Play Function

Start playback.

File

drv_bm64.h

C

```
void DRV_BM64_Play(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_Play:

DRV_BM64_Play(const DRV_HANDLE handle);

Send an AVRCP command to the host device to initiate or resume playback.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_Play(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_PlayNextSong Function

Play the next song.

File

drv_bm64.h

C

```
void DRV_BM64_PlayNextSong(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_PlayNextSong:

void DRV_BM64_PlayNextSong(const DRV_HANDLE handle);

Send an AVRCP command to the host device to play the next song in a playlist.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_PlayNextSong(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_PlayPause Function

Toggle play/pause mode.

File

drv_bm64.h

C

```
void DRV_BM64_PlayPause(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_PlayPause:

void DRV_BM64_PlayPause(const DRV_HANDLE handle);

Send an AVRCP command to the host device to toggle the play/pause mode.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_PlayPause(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_PlayPreviousSong Function

Play the previous song.

File

drv_bm64.h

C

```
void DRV_BM64_PlayPreviousSong(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_PlayPreviousSong:

void DRV_BM64_PlayPreviousSong(const DRV_HANDLE handle);

Send an AVRCP command to the host device to play the previous song in a playlist.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_5)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_PlayPreviousSong(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_Rewind Function

Rewind the media.

File

drv_bm64.h

C

```
void DRV_BM64_Rewind(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_Rewind:

void DRV_BM64_Rewind(const DRV_HANDLE handle);

Send an AVRCP command to the host device to rewind the media.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_5)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_Rewind(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_Stop Function

Stop playback.

File

drv_bm64.h

C

```
void DRV_BM64_Stop(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_Stop:

void DRV_BM64_Stop(const DRV_HANDLE handle);

Send an AVRCP command to the host device to stop playback.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
case BUTTON_STATE_PRESSED:      // (debouncing not shown)
{
    if (BSP_SwitchStateGet(BSP_SWITCH_3)==BSP_SWITCH_STATE_PRESSED))
    {
        // note generic version of call (DRV_BT instead of DRV_BM64) is used
        DRV_BT_Stop(appData.bt.handle);
        appData.buttonState=BUTTON_STATE_WAIT_FOR_RELEASE;
    }
}
break;
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

g) Device Name and Address Functions

DRV_BM64_GetBDAddress Function

Return the Bluetooth address.

File

drv_bm64.h

C

```
void DRV_BM64_GetBDAddress(const DRV_HANDLE handle, char* buffer);
```

Returns

None.

Description

Function DRV_BM64_GetBDAddress:

void DRV_BM64_GetBDAddress(const DRV_HANDLE handle, char* buffer);

Return the Bluetooth address of the device as an ASCII string.

Remarks

Buffer must be at least 18 bytes in length (6 octets separated by ?:?, e.g. able to hold "12:34:56:78:90:120").

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
laString tempStr;
char buf [18];

// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_GetBDAddress(appData.bt.handle, buf);
tempStr = laString_CreateFromCharBuffer(buf, &LiberationSans12);
laLabelWidget_SetText(GFX_BTADDRESS_VALUE, tempStr);    // display BT address
laString_Destroy(&tempStr);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
buffer	pointer to a char buffer at least 18 bytes long

DRV_BM64_GetBDName Function

Return Bluetooth device name.

File

drv_bm64.h

C

```
void DRV_BM64_GetBDName(const DRV_HANDLE handle, char* buffer, const uint8_t buflen);
```

Returns

None.

Description

Function DRV_BM64_GetBDName:

void DRV_BM64_GetBDName(const DRV_HANDLE handle, char* buffer, const uint8_t buflen);

Return the Bluetooth device name as an ASCII string.

Remarks

If name is longer than buflen-1 bytes long, it will be truncated to fit inside the buffer.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
laString tempStr;
char buf [DRV_BT_MAXBDNAMESIZE+1];

// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_GetBDName(appData.bt.handle, buf, DRV_BT_MAXBDNAMESIZE+1);
tempStr = laString_CreateFromCharBuffer(buf, &LiberationSans12);
laLabelWidget_SetText(GFX_BTNAME_VALUE, tempStr);   // display BT name
laString_Destroy(&tempStr);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
buffer	pointer to a char buffer at least buflen bytes long
buflen	length of buffer (including terminating 0 byte)

DRV_BM64_SetBDName Function

Set the Bluetooth device name.

File

drv_bm64.h

C

```
void DRV_BM64_SetBDName(const DRV_HANDLE handle, const char* buffer);
```

Returns

None.

Description

Function DRV_BM64_SetBDName:

void DRV_BM64_SetBTName(const DRV_HANDLE handle, const char* buffer);

Set a temporary Bluetooth device name from an ASCII string buffer.

Remarks

The name is set for this session only; if the BM64 is reset (e.g. power is lost) the name will revert to the Bluetooth name stored in EEPROM.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_SetBDName(appData.bt.handle, "Temporary BM64 Name");
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
buffer	pointer to a char buffer containing the new name

h) BLE Functions

DRV_BM64_ClearBLEData Function

Clear the BLE receive buffer.

File

drv_bm64.h

C

```
void DRV_BM64_ClearBLEData(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_ClearBLEData:

void DRV_BM64_ClearBLEData(const DRV_HANDLE handle);

Clears the buffer used when receiving characters via the DRV_BM64_ReadByteFromBLE and DRV_BM64_ReadDataFromBLE calls.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
uint8_t byte;

// note generic versions of calls (DRV_BT instead of DRV_BM64) is used
DRV_BT_ClearBLEData(appData.bt.handle);

// wait for byte to arrive
while (!DRV_BT_ReadByteFromBLE(appData.bt.handle, &byte))
{
    // should have some sort of way to break out of here if byte never arrives
}
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_ReadByteFromBLE Function

Read a byte over BLE.

File

drv_bm64.h

C

```
bool DRV_BM64_ReadByteFromBLE(const DRV_HANDLE handle, uint8_t* byte);
```

Returns

bool - true if a byte was returned, false if receive buffer empty

Description

Function DRV_BM64_ReadByteFromBLE:

bool DRV_BM64_ReadByteFromBLE(const DRV_HANDLE handle, uint8_t* byte);

Read one byte over BLE using the BM64's "Transparent Service" feature.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
uint8_t byte;

// note generic version of call (DRV_BT instead of DRV_BM64) is used
if (DRV_BT_ReadByteFromBLE(appData.bt.handle, &byte))  // if byte received
{
    // do something
}
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
byte	pointer to a uint8_t to receive the data

DRV_BM64_ReadDataFromBLE Function

Read data over BLE.

File

drv_bm64.h

C

```
bool DRV_BM64_ReadDataFromBLE(const DRV_HANDLE handle, uint8_t* byte, uint16_t size);
```

Returns

bool - true if data was returned, false if receive buffer empty

Description

Function DRV_BM64_ReadDataFromBLE:

bool DRV_BM64_ReadDataFromBLE(const DRV_HANDLE handle, uint8_t* bytes, uint16_t size);

Read data over BLE using the BM64's "Transparent Service" feature.

Remarks

No more than size bytes will be returned, even if more are available.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
#define BUFSIZE  100
uint8_t buf [BUFSIZE];

// note generic version of call (DRV_BT instead of DRV_BM64) is used
if (DRV_BT_ReadDataFromBLE(appData.bt.handle, buf, BUFSIZE))  // if data received
```

```
{
    // do something
}
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
bytes	pointer to a uint8_t buffer at least size bytes long
size	length of buffer (including

DRV_BM64_SendByteOverBLE Function

Send a byte over BLE.

File

drv_bm64.h

C

```
void DRV_BM64_SendByteOverBLE(const DRV_HANDLE handle, uint8_t byte);
```

Returns

None.

Description

Function DRV_BM64_SendByteOverBLE:

void DRV_BM64_SendByteOverBLE(const DRV_HANDLE handle, uint8_t byte);

Send one byte over BLE using the BM64's "Transparent Service" feature.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
uint8_t byte;

byte = 10;     // set to some value

// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_SendByteOverBLE(appData.bt.handle, byte);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
byte	uint8_t of data to be sent

DRV_BM64_SendDataOverBLE Function

Send data over BLE.

File

drv_bm64.h

C

```
void DRV_BM64_SendDataOverBLE(const DRV_HANDLE handle, uint8_t* bytes, uint16_t size);
```

Returns

None.

Description

Function DRV_BM64_SendDataOverBLE:

void DRV_BM64_SendDataOverBLE(const DRV_HANDLE handle, uint8_t* bytes, uint16_t size);

Send data over BLE using the BM64's "Transparent Service" feature.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
#define BUFSIZE    100
uint8_t buf [BUFSIZE];

// (code to fill in buffer with data)

// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_SendDataOverBLE(appData.bt.handle, buf, BUFSIZE);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
bytes	pointer to a uint8_t buffer at least size bytes long
size	length of buffer (including

DRV_BM64_BLE_QueryStatus Function

Query BM64 LE status.

File

drv_bm64.h

C

```
void DRV_BM64_BLE_QueryStatus(const DRV_HANDLE handle);
```

Returns

None.

Description

Function DRV_BM64_BLE_QueryStatus:

void DRV_BM64_BLE_QueryStatus(const DRV_HANDLE handle);

Queries the BM64 to respond with a DRV_BM64_EVENT_BLE_STATUS_CHANGED event, which will indicate if the BM64 BLE status is standby, advertising, scanning or connected.

Remarks

RV_BM64_BLE_QueryStatus is non-blocking; it returns right away and sometime later (perhaps tens or hundreds of ms) the event handler callback will be called.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BT_BLE_QueryStatus(appData.bt.handle);

. . .

// later, a call will come back to the event handler callback function
// (previously set up via a call to DRV_BM64_EventHandlerSet)
static void _BLEEventHandler(DRV_BT_EVENT event, uint32_t param, uintptr_t context)
```

```
{
    switch(event)
    {
        case DRV_BT_EVENT_BLE_STATUS_CHANGED:
        {
            // do case switch based on param variable
        }
   }
}
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client

DRV_BM64_BLE_EnableAdvertising Function

Enable or disable advertising.

File

drv_bm64.h

C

```
void DRV_BM64_BLE_EnableAdvertising(const DRV_HANDLE handle, bool enable);
```

Returns

None.

Description

Function DRV_BM64_BLE_EnableAdvertising:

void DRV_BM64_BLE_EnableAdvertising(const DRV_HANDLE handle, bool enable);

Enable or disable BLE advertising.

Remarks

None.

Preconditions

DRV_BM64_Open must have been called to obtain a valid opened device handle.

Example

```
// note generic version of call (DRV_BT instead of DRV_BM64) is used
DRV_BM64_BLE_EnableAdvertising(appData.bt.handle, true);
```

Parameters

Parameters	Description
handle	valid handle to an opened BM64 device driver unique to client
enable	true to enable advertising, false to disable advertising

i) Data Types and Constants

DRV_BM64_BUFFER_EVENT Macro

File

drv_bm64.h

C

```
#define DRV_BM64_BUFFER_EVENT DRV_I2S_BUFFER_EVENT
```

Description

This is macro DRV_BM64_BUFFER_EVENT.

DRV_BM64_BUFFER_EVENT_COMPLETE Macro

File

drv_bm64.h

C

```
#define DRV_BM64_BUFFER_EVENT_COMPLETE DRV_I2S_BUFFER_EVENT_COMPLETE
```

Description

This is macro DRV_BM64_BUFFER_EVENT_COMPLETE.

DRV_BM64_BUFFER_HANDLE Macro

File

drv_bm64.h

C

```
#define DRV_BM64_BUFFER_HANDLE DRV_I2S_BUFFER_HANDLE
```

Description

This is macro DRV_BM64_BUFFER_HANDLE.

DRV_BM64_BUFFER_HANDLE_INVALID Macro

File

drv_bm64.h

C

```
#define DRV_BM64_BUFFER_HANDLE_INVALID DRV_I2S_BUFFER_HANDLE_INVALID
```

Description

This is macro DRV_BM64_BUFFER_HANDLE_INVALID.

DRV_BM64_DATA32 Macro

File

drv_bm64.h

C

```
#define DRV_BM64_DATA32 DRV_I2S_DATA32
```

Description

BM64 defines based on I2S interface

DRV_BM64_MAXBDNAMESIZE Macro

File

drv_bm64.h

C

```
#define DRV_BM64_MAXBDNAMESIZE 32
```

Section

Constants

DRV_BM64_BUFFER_EVENT_HANDLER Type

File

drv_bm64.h

C

```
typedef void (* DRV_BM64_BUFFER_EVENT_HANDLER)(DRV_BM64_BUFFER_EVENT event, uintptr_t contextHandle);
```

Description

prototype for callback for DRV_BM64_BufferEventHandlerSet

DRV_BM64_DRVR_STATUS Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_STATUS_NONE,
  DRV_BM64_STATUS_OFF,
  DRV_BM64_STATUS_ON,
  DRV_BM64_STATUS_READY
} DRV_BM64_DRVR_STATUS;
```

Description

BM64 driver status

DRV_BM64_EVENT Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_EVENT_NONE = 0,
  DRV_BM64_EVENT_NSPK_STATUS,
  DRV_BM64_EVENT_LINE_IN_STATUS,
  DRV_BM64_EVENT_A2DP_STATUS,
  DRV_BM64_EVENT_CALL_STATUS_CHANGED,
  DRV_BM64_EVENT_CODEC_TYPE,
  DRV_BM64_EVENT_HFP_CONNECTED,
  DRV_BM64_EVENT_HFP_DISCONNECTED,
  DRV_BM64_EVENT_A2DP_CONNECTED,
  DRV_BM64_EVENT_A2DP_DISCONNECTED,
  DRV_BM64_EVENT_AVRCP_CONNECTED,
  DRV_BM64_EVENT_AVRCP_DISCONNECTED,
  DRV_BM64_EVENT_SPP_CONNECTED,
  DRV_BM64_EVENT_IAP_CONNETED,
  DRV_BM64_EVENT_SPP_IAP_DISCONNECTED,
  DRV_BM64_EVENT_ACL_CONNECTED,
  DRV_BM64_EVENT_ACL_DISCONNECTED,
  DRV_BM64_EVENT_SCO_CONNECTED,
  DRV_BM64_EVENT_SCO_DISCONNECTED,
  DRV_BM64_EVENT_MAP_CONNECTED,
  DRV_BM64_EVENT_MAP_DISCONNECTED,
  DRV_BM64_EVENT_SYS_POWER_ON,
  DRV_BM64_EVENT_SYS_POWER_OFF,
  DRV_BM64_EVENT_SYS_STANDBY,
  DRV_BM64_EVENT_SYS_PAIRING_START,
  DRV_BM64_EVENT_SYS_PAIRING_OK,
  DRV_BM64_EVENT_SYS_PAIRING_FAILED,
  DRV_BM64_EVENT_LINKBACK_SUCCESS,
  DRV_BM64_EVENT_LINKBACK_FAILED,
  DRV_BM64_EVENT_BD_ADDR_RECEIVED,
  DRV_BM64_EVENT_PAIR_RECORD_RECEIVED,
  DRV_BM64_EVENT_LINK_MODE_RECEIVED,
```

```
    DRV_BM64_EVENT_PLAYBACK_STATUS_CHANGED,
    DRV_BM64_EVENT_AVRCP_VOLUME_CTRL,
    DRV_BM64_EVENT_AVRCP_ABS_VOLUME_CHANGED,
    DRV_BM64_EVENT_HFP_VOLUME_CHANGED,
    DRV_BM64_EVENT_VOLUME_CHANGED,
    DRV_BM64_EVENT_SAMPLERATE_CHANGED,
    DRV_BM64_EVENT_NSPK_SYNC_POWER_OFF,
    DRV_BM64_EVENT_NSPK_SYNC_VOL_CTRL,
    DRV_BM64_EVENT_NSPK_SYNC_INTERNAL_GAIN,
    DRV_BM64_EVENT_NSPK_SYNC_ABS_VOL,
    DRV_BM64_EVENT_NSPK_CHANNEL_SETTING,
    DRV_BM64_EVENT_NSPK_ADD_SPEAKER3,
    DRV_BM64_EVENT_LE_STATUS_CHANGED,
    DRV_BM64_EVENT_LE_ADV_CONTROL_REPORT,
    DRV_BM64_EVENT_LE_CONNECTION_PARA_REPORT,
    DRV_BM64_EVENT_LE_CONNECTION_PARA_UPDATE_RSP,
    DRV_BM64_EVENT_GATT_ATTRIBUTE_DATA,
    DRV_BM64_EVENT_PORT0_INPUT_CHANGED,
    DRV_BM64_EVENT_PORT1_INPUT_CHANGED,
    DRV_BM64_EVENT_PORT2_INPUT_CHANGED,
    DRV_BM64_EVENT_PORT3_INPUT_CHANGED,
    DRV_BM64_EVENT_BLESPP_MSG_RECEIVED,
    DRV_BM64_EVENT_BLE_STATUS_CHANGED
} DRV_BM64_EVENT;
```

Description

events that can be returned to a client via callback

DRV_BM64_EVENT_HANDLER Type

File

drv_bm64.h

C

```
typedef void (* DRV_BM64_EVENT_HANDLER)(DRV_BM64_EVENT event, uint32_t param, uintptr_t contextHandle);
```

Description

prototype for callback for DRV_BM64_EventHandlerSet

DRV_BM64_LINKSTATUS Enumeration

File

drv_bm64.h

C

```
typedef enum {
    DRV_BM64_NO_LINK_STATUS = 0,
    DRV_BM64_SCO_LINK_STATUS = 0x01,
    DRV_BM64_ACL_LINK_STATUS = 0x02,
    DRV_BM64_HFP_LINK_STATUS = 0x04,
    DRV_BM64_A2DP_LINK_STATUS = 0x08,
    DRV_BM64_AVRCP_LINK_STATUS = 0x10,
    DRV_BM64_SPP_LINK_STATUS = 0x20,
    DRV_BM64_IAP_LINK_STATUS = 0x40,
    DRV_BM64_MAP_LINK_STATUS = 0x80
} DRV_BM64_LINKSTATUS;
```

Description

BM64 link status

DRV_BM64_PLAYINGSTATUS Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_PLAYING_STOPPED,
  DRV_BM64_PLAYING_PLAYING,
  DRV_BM64_PLAYING_PAUSED,
  DRV_BM64_PLAYING_FF,
  DRV_BM64_PLAYING_FR,
  DRV_BM64_PLAYING_ERROR
} DRV_BM64_PLAYINGSTATUS;
```

Description

This is type DRV_BM64_PLAYINGSTATUS.

DRV_BM64_PROTOCOL Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_PROTOCOL_A2DP = 1,
  DRV_BM64_PROTOCOL_AVRCP = 2,
  DRV_BM64_PROTOCOL_HFP_HSP = 4,
  DRV_BM64_PROTOCOL_SPP = 8,
  DRV_BM64_PROTOCOL_BLE = 16,
  DRV_BM64_PROTOCOL_ALL = 31
} DRV_BM64_PROTOCOL;
```

Description

BM64 protocols

DRV_BM64_REQUEST Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_REQ_NONE = 0,
  DRV_BM64_REQ_SYSTEM_ON,
  DRV_BM64_REQ_SYSTEM_OFF
} DRV_BM64_REQUEST;
```

Description

BM64 power on/off request

DRV_BM64_SAMPLE_FREQUENCY Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_SAMPLEFREQ_8000 = 0,
  DRV_BM64_SAMPLEFREQ_12000,
  DRV_BM64_SAMPLEFREQ_16000,
  DRV_BM64_SAMPLEFREQ_24000,
  DRV_BM64_SAMPLEFREQ_32000,
  DRV_BM64_SAMPLEFREQ_48000,
  DRV_BM64_SAMPLEFREQ_44100,
  DRV_BM64_SAMPLEFREQ_88000,
  DRV_BM64_SAMPLEFREQ_96000
} DRV_BM64_SAMPLE_FREQUENCY;
```

Description

BM64 sample frequency

DRV_BM64_BLE_STATUS Enumeration

File

drv_bm64.h

C

```
typedef enum {
  DRV_BM64_BLE_STATUS_STANDBY,
  DRV_BM64_BLE_STATUS_ADVERTISING,
  DRV_BM64_BLE_STATUS_SCANNING,
  DRV_BM64_BLE_STATUS_CONNECTED
} DRV_BM64_BLE_STATUS;
```

Description

This is type DRV_BM64_BLE_STATUS.

Files

Files

Name	Description
drv_bm64.h	BM64 Bluetooth Static Driver main header file
drv_bm64_config_template.h	BM64 Bluetooth Driver Configuration Template.

Description

This section lists the source and header files used by the BM64 Bluetooth Driver Library.

drv_bm64.h

BM64 Bluetooth Static Driver main header file

Enumerations

	Name	Description
	DRV_BM64_BLE_STATUS	This is type DRV_BM64_BLE_STATUS.
	DRV_BM64_DRVR_STATUS	BM64 driver status
	DRV_BM64_EVENT	events that can be returned to a client via callback
	DRV_BM64_LINKSTATUS	BM64 link status
	DRV_BM64_PLAYINGSTATUS	This is type DRV_BM64_PLAYINGSTATUS.
	DRV_BM64_PROTOCOL	BM64 protocols
	DRV_BM64_REQUEST	BM64 power on/off request
	DRV_BM64_SAMPLE_FREQUENCY	BM64 sample frequency

Functions

	Name	Description
	DRV_BM64_BLE_EnableAdvertising	Enable or disable advertising.
	DRV_BM64_BLE_QueryStatus	Query BM64 LE status.
	DRV_BM64_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_BM64_BufferEventHandlerSet	This function allows a client to identify a event handling function for the driver to call back.
	DRV_BM64_CancelForwardOrRewind	Cancel previous fast forward or rewind request.
	DRV_BM64_ClearBLEData	Clear the BLE receive buffer.
	DRV_BM64_Close	Close an opened-instance of the BM64 Bluetooth driver.
	DRV_BM64_DisconnectAllLinks	Disconnect all links.
	DRV_BM64_EnterBTPairingMode	Enter Bluetooth pairing mode.
	DRV_BM64_EventHandlerSet	This function allows a client to identify an event handling function for the driver to call back.
	DRV_BM64_FastForward	Fast forward the media.

	DRV_BM64_ForgetAllLinks	Forget all pairings.
	DRV_BM64_GetBDAddress	Return the Bluetooth address.
	DRV_BM64_GetBDName	Return Bluetooth device name.
	DRV_BM64_GetLinkStatus	Return link status.
	DRV_BM64_GetPlayingStatus	Return the current playing status of the device.
	DRV_BM64_GetPowerStatus	Gets the current status of the BM64 Bluetooth driver module (BM64-specific).
	DRV_BM64_Initialize	Initializes hardware and data for the instance of the BM64 Bluetooth module
	DRV_BM64_LinkLastDevice	Link last device.
	DRV_BM64_Open	Open the specified BM64 driver instance and returns a handle to it
	DRV_BM64_Pause	Pause playback.
	DRV_BM64_Play	Start playback.
	DRV_BM64_PlayNextSong	Play the next song.
	DRV_BM64_PlayPause	Toggle play/pause mode.
	DRV_BM64_PlayPreviousSong	Play the previous song.
	DRV_BM64_ReadByteFromBLE	Read a byte over BLE.
	DRV_BM64_ReadDataFromBLE	Read data over BLE.
	DRV_BM64_Rewind	Rewind the media.
	DRV_BM64_SamplingRateGet	Return the current sampling rate.
	DRV_BM64_SamplingRateSet	Set the current sampling rate.
	DRV_BM64_SendByteOverBLE	Send a byte over BLE.
	DRV_BM64_SendDataOverBLE	Send data over BLE.
	DRV_BM64_SetBDName	Set the Bluetooth device name.
	DRV_BM64_Status	Gets the current system status of the BM64 Bluetooth driver module.
	DRV_BM64_Stop	Stop playback.
	DRV_BM64_TaskReq	Make a power on/power off task request.
	DRV_BM64_Tasks	Maintains the driver's control and data interface state machine.
	DRV_BM64_volumeDown	Turn the volume down on the host device.
	DRV_BM64_VolumeGet	returns 7-bit value 0-127
	DRV_BM64_VolumeSet	Set current volume.
	DRV_BM64_volumeUp	Turn the volume up on the host device.

Macros

	Name	Description
	DRV_BM64_BUFFER_EVENT	This is macro DRV_BM64_BUFFER_EVENT.
	DRV_BM64_BUFFER_EVENT_COMPLETE	This is macro DRV_BM64_BUFFER_EVENT_COMPLETE.
	DRV_BM64_BUFFER_HANDLE	This is macro DRV_BM64_BUFFER_HANDLE.
	DRV_BM64_BUFFER_HANDLE_INVALID	This is macro DRV_BM64_BUFFER_HANDLE_INVALID.
	DRV_BM64_DATA32	BM64 defines based on I2S interface
	DRV_BM64_MAXBDNAMESIZE	

Types

	Name	Description
	DRV_BM64_BUFFER_EVENT_HANDLER	prototype for callback for DRV_BM64_BufferEventHandlerSet
	DRV_BM64_EVENT_HANDLER	prototype for callback for DRV_BM64_EventHandlerSet

Description

BM64 Bluetooth Static Driver implementation

This file is the header file for the external (public) API of the static implementation of the BM64 driver.

The BM64 is a Bluetooth 4.2 Stereo Module that supports classic A2DP, AVRCP, HFP, HSP, and SPP protocols as well as BLE (Bluetooth Low Energy).

The BM64 streams I2S audio at up to 24-bit, 96 kHz. It uses a UART to receive commands from the host microcontroller (PIC32) and and send events back.

All functions and constants in this file are named with the format DRV_BM64_xxx, where xxx is a function name or constant. These names are redefined in the appropriate configuration?s system_config.h file to the format DRV_BT_xxx using #defines so that Bluetooth code in the application can be written as generically as possible (e.g. by writing DRV_BT_Open instead of DRV_BM64_Open etc.).

File Name

drv_bm64.h

Company

Microchip Technology Inc.

drv_bm64_config_template.h

BM64 Bluetooth Driver Configuration Template.

Macros

	Name	Description
	INCLUDE_BM64_BLE	Identifies whether the driver should include BLE
	INCLUDE_BM64_I2S	Identifies whether the driver should include HFP,A2DP,AVRCP functionality.
	INCLUDE_DEPRECATED_MMI_COMMANDS	Identifies whether the driver should use deprecated MMI commands.

Description

BM64 Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_bm64_config_template.h

Company

Microchip Technology Inc.

Camera Driver Libraries

This section describes the Camera Driver Libraries.

Introduction

This section provides information on the Camera Driver libraries that are provided in MPLAB Harmony and describes the APIs that are common to all drivers.

Library Interface

a) Common Driver Functions

	Name	Description
	DRV_CAMERA_Close	Closes an opened instance of an CAMERA module driver.
	DRV_CAMERA_Deinitialize	Deinitializes the index instance of the CAMERA module.
	DRV_CAMERA_Initialize	Initializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Open	Opens the specified instance of the Camera driver for use and provides an "open instance" handle.
	DRV_CAMERA_Reinitialize	Reinitializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Status	Provides the current status of the index instance of the CAMERA module.
	DRV_CAMERA_Tasks	This is function DRV_CAMERA_Tasks.

b) Common Data Types and Constants

	Name	Description
	DRV_CAMERA_INIT	Defines the data required to initialize or reinitialize the CAMERA driver.
	DRV_CAMERA_INTERRUPT_PORT_REMAP	Defines the data required to initialize the CAMERA driver interrupt port remap.
	DRV_CAMERA_INDEX_0	Camera driver index definitions.
	DRV_CAMERA_INDEX_1	This is macro DRV_CAMERA_INDEX_1.
	DRV_CAMERA_INDEX_COUNT	Number of valid CAMERA driver indices.
	CAMERA_MODULE_ID	This is type CAMERA_MODULE_ID.

Description

Camera Driver APIs that are common to all Camera drivers.

a) Common Driver Functions

DRV_CAMERA_Close Function

Closes an opened instance of an CAMERA module driver.

File

drv_camera.h

C

```
void DRV_CAMERA_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of an CAMERA module driver, making the specified handle invalid.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize routine must have been called for the specified CAMERA device instance and the DRV_CAMERA_Status must have returned SYS_STATUS_READY.

DRV_CAMERA_Open must have been called to obtain a valid opened device handle.

Example

```
myCameraHandle = DRV_CAMERA_Open(DRV_CAMERA_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

DRV_CAMERA_Close(myCameraHandle);
```

Parameters

Parameters	Description
drvHandle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_CAMERA_Close (const DRV_HANDLE drvHandle)

DRV_CAMERA_Deinitialize Function

Deinitializes the index instance of the CAMERA module.

File

drv_camera.h

C

```
void DRV_CAMERA_Deinitialize(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

This function deinitializes the index instance of the CAMERA module, disabling its operation (and any hardware for driver modules). It deinitializes only the specified module instance. It also resets all the internal data structures and fields for the specified instance to the default settings.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS   cameraStatus;

DRV_CAMERA_Deinitialize(DRV_CAMERA_ID_1);

cameraStatus = DRV_CAMERA_Status(DRV_CAMERA_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be deinitialized

Function

void DRV_CAMERA_Deinitialize (const SYS_MODULE_ID index)

DRV_CAMERA_Initialize Function

Initializes hardware and data for the index instance of the CAMERA module.

File

drv_camera.h

C

```
SYS_MODULE_OBJ DRV_CAMERA_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function initializes hardware for the index instance of the CAMERA module, using the hardware initialization given data. It also initializes any internal driver data structures making the driver ready to be opened.

Remarks

None.

Preconditions

None.

Example

```
DRV_CAMERA_INIT_DATA        cameraInitData;
SYS_STATUS                  cameraStatus;

// Populate the cameraInitData structure
cameraInitData.moduleInit.powerState = SYS_MODULE_POWER_RUN_FULL;
cameraInitData.moduleInit.moduleCode = (DRV_CAMERA_INIT_DATA_MASTER | DRV_CAMERA_INIT_DATA_SLAVE);

DRV_CAMERA_Initialize(DRV_CAMERA_ID_1, (SYS_MODULE_INIT*)&cameraInitData);
cameraStatus = DRV_CAMERA_Status(DRV_CAMERA_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be initialized
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and the default initialization is to be used.

Function

void DRV_CAMERA_Initialize (const CAMERA_MODULE_ID index,

const SYS_MODULE_INIT *const data)

DRV_CAMERA_Open Function

Opens the specified instance of the Camera driver for use and provides an "open instance" handle.

File

drv_camera.h

C

```
DRV_HANDLE DRV_CAMERA_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a value identifying both the caller and the module instance). If an error occurs, the returned value is DRV_HANDLE_INVALID.

Description

This function opens the specified instance of the Camera module for use and provides a handle that is required to use the remaining driver routines.

This function opens a specified instance of the Camera module driver for use by any client module and provides an "open instance" handle that must be provided to any of the other Camera driver operations to identify the caller and the instance of the Camera driver/hardware module.

Preconditions

The DRV_CAMERA_Initialize routine must have been called for the specified CAMERA device instance and the DRV_CAMERA_Status must have returned SYS_STATUS_READY.

Example

```
DRV_HANDLE                cameraHandle;
DRV_CAMERA_CLIENT_STATUS   cameraClientStatus;

cameraHandle = DRV_CAMERA_Open(DRV_CAMERA_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == cameraHandle)
{
    // Handle open error
}

cameraClientStatus = DRV_CAMERA_ClientStatus(cameraHandle);

// Close the device when it is no longer needed.
DRV_CAMERA_Close(cameraHandle);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be opened.
intent	Flags parameter identifying the intended usage and behavior of the driver. Multiple flags may be ORed together to specify the intended usage of the device. See the DRV_IO_INTENT definition.

Function

DRV_HANDLE DRV_CAMERA_Open (const SYS_MODULE_INDEX index,
const DRV_IO_INTENT intent)

DRV_CAMERA_Reinitialize Function

Reinitializes hardware and data for the index instance of the CAMERA module.

File

drv_camera.h

C

```
void DRV_CAMERA_Reinitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT *const data);
```

Returns

None.

Description

This function reinitializes hardware for the index instance of the CAMERA module, using the hardware initialization given data. It also reinitializes any internal driver data structures making the driver ready to be opened.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_INIT cameraInit;
SYS_STATUS      cameraStatus;

DRV_CAMERA_Reinitialize(DRV_CAMERA_ID_1, &cameraStatus);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be reinitialized
data	Pointer to the data structure containing any data necessary to reinitialize the hardware. This pointer may be null if no data is required and default configuration is to be used.

Function

void DRV_CAMERA_Reinitialize(const SYS_MODULE_ID index,

const SYS_MODULE_INIT *const data)

DRV_CAMERA_Status Function

Provides the current status of the index instance of the CAMERA module.

File

drv_camera.h

C

```
SYS_STATUS DRV_CAMERA_Status(const SYS_MODULE_INDEX index);
```

Returns

The current status of the index instance.

Description

This function provides the current status of the index instance of the CAMERA module.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Function

SYS_STATUS DRV_CAMERA_Status (const CAMERA_MODULE_ID index)

DRV_CAMERA_Tasks Function

File

drv_camera.h

C

```
void DRV_CAMERA_Tasks(SYS_MODULE_OBJ object);
```

Description

This is function DRV_CAMERA_Tasks.

b) Common Data Types and Constants

DRV_CAMERA_INIT Structure

Defines the data required to initialize or reinitialize the CAMERA driver.

File

drv_camera.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  int cameraId;
  SYS_MODULE_OBJ (* drvInitialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
  DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  INT_SOURCE interruptSource;
  DRV_CAMERA_INTERRUPT_PORT_REMAP interruptPort;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
} DRV_CAMERA_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
int cameraId;	ID
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels

Description

CAMERA Driver Initialization Data

This data type defines the data required to initialize or reinitialize the CAMERA driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_CAMERA_INTERRUPT_PORT_REMAP Structure

Defines the data required to initialize the CAMERA driver interrupt port remap.

File

drv_camera.h

C

```
typedef struct {
  PORTS_REMAP_INPUT_FUNCTION inputFunction;
  PORTS_REMAP_INPUT_PIN inputPin;
  PORTS_ANALOG_PIN analogPin;
  PORTS_PIN_MODE pinMode;
  PORTS_CHANNEL channel;
  PORTS_DATA_MASK dataMask;
} DRV_CAMERA_INTERRUPT_PORT_REMAP;
```

Description

CAMERA Driver Interrupt Port Remap Initialization Data

This data type defines the data required to initialize the CAMERA driver interrupt port remap.

Remarks

None.

DRV_CAMERA_INDEX_0 Macro

Camera driver index definitions.

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_0 0
```

Description

Camera Driver Module Index Numbers

These constants provide the Camera driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_CAMERA_Initialize and DRV_CAMERA_Open functions to identify the driver instance in use.

DRV_CAMERA_INDEX_1 Macro

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_1 1
```

Description

This is macro DRV_CAMERA_INDEX_1.

DRV_CAMERA_INDEX_COUNT Macro

Number of valid CAMERA driver indices.

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_COUNT 1
```

Description

CAMERA Driver Module Index Count

This constant identifies the number of valid CAMERA driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

CAMERA_MODULE_ID Enumeration

File

drv_camera.h

C

```
typedef enum {
  CAMERA_MODULE_OVM7690
} CAMERA_MODULE_ID;
```

Description

This is type CAMERA_MODULE_ID.

Files

Files

Name	Description
drv_camera.h	Camera device driver interface file.

Description

drv_camera.h

Camera device driver interface file.

Enumerations

	Name	Description
	CAMERA_MODULE_ID	This is type CAMERA_MODULE_ID.

Functions

	Name	Description
	DRV_CAMERA_Close	Closes an opened instance of an CAMERA module driver.
	DRV_CAMERA_Deinitialize	Deinitializes the index instance of the CAMERA module.
	DRV_CAMERA_Initialize	Initializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Open	Opens the specified instance of the Camera driver for use and provides an "open instance" handle.
	DRV_CAMERA_Reinitialize	Reinitializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Status	Provides the current status of the index instance of the CAMERA module.
	DRV_CAMERA_Tasks	This is function DRV_CAMERA_Tasks.

Macros

	Name	Description
	DRV_CAMERA_INDEX_0	Camera driver index definitions.
	DRV_CAMERA_INDEX_1	This is macro DRV_CAMERA_INDEX_1.
	DRV_CAMERA_INDEX_COUNT	Number of valid CAMERA driver indices.

Structures

	Name	Description
	DRV_CAMERA_INIT	Defines the data required to initialize or reinitialize the CAMERA driver.
	DRV_CAMERA_INTERRUPT_PORT_REMAP	Defines the data required to initialize the CAMERA driver interrupt port remap.

Description

Camera Driver Interface

The Camera driver provides a abstraction to all camera drivers.

File Name

drv_camera.h

Company

Microchip Technology Inc.

OVM7690 Camera Driver Library

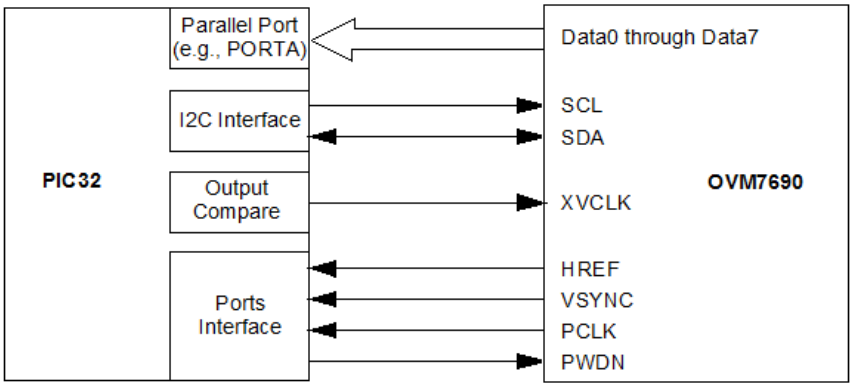
This topic describes the OVM7690 Camera Driver Library.

Introduction

The OVM7690 Camera Driver provides a high-level interface to manage the OmniVision Technologies, Inc. OVM7690 640x480 CameraCube™ device (referred to as the OVM7690) that is interfaced with serial and parallel ports to a Microchip microcontroller for providing camera solutions.

Description

The OVM7690 640x480 CameraCube™ device (referred to as the OVM7690) can be interfaced to a Microchip microcontroller using the I2C serial interface and parallel port interface. The I2C serial interface is used for control command transfer. The I2C module from the microcontroller is connected to the SCCB serial interface of the OVM7690. The parallel port interface is used to transfer pixel data from the OVM7690 to the microcontroller. There are few other signals from the camera to be interfaced with the microcontroller. The XVCLK pin of the camera is driven by the Output Compare module. Frame synchronization signals such as HREF and VSYNC from the camera are connected to suitable pins supporting change notification within the microcontroller. The PCLK pin of the camera drives the pixel clock and is connected at the pin of the microcontroller supporting external interrupts. The PWDN pin of the camera supports camera power-down mode and is connected at any output port pin of the microcontroller. A typical interface of the OVM7690 to a PIC32 device is provided in the following diagram:



Using the Library

This topic describes the basic architecture of the OVM7690 Camera Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_camera_ovm7690.h`

The interface to the Camera Driver Library is defined in the `drv_camera_ovm7690.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address the overall operation of the OVM7690 Camera Driver Library.

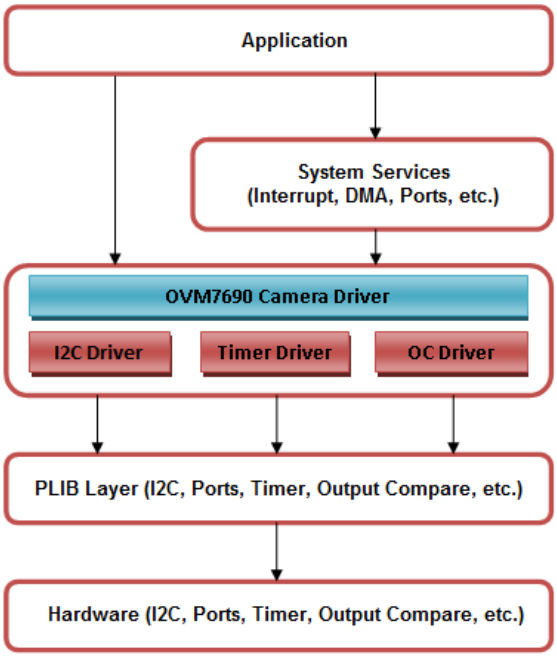
Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization and deinitialization.
Client Setup Functions	Provides open and close functions.
Camera-specific Functions	Provides APIs that are camera-specific.
Other Functions	Provides miscellaneous driver-specific functions such as register set functions, among others.

Abstraction Model

This library provides a low-level abstraction of the OVM7690 Camera Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The OVM7690 Camera Driver is modeled using the abstraction model, as shown in the following diagram.



How the Library Works

Provides information on how the OVM7690 Camera Driver Library works.

Description

The library provides interfaces to support:

- System functionality
- Client functionality

System Initialization

The system performs the Initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the OVM7690 would be initialized with the following configuration settings that are supported by the specific OVM7690 device hardware:

- Camera ID: OVM7690 ID
- Source Port: Address of source port to which the pixel data is received
- Horizontal Sync Channel: Channel of the pin to be configured as horizontal sync
- Horizontal Sync Position: Horizontal sync port pin position from selected port channel
- Vertical Sync Channel: Channel the pin to be configured as vertical sync
- Vertical Sync Position: Vertical sync port pin position from selected port channel
- Horizontal Sync Interrupt Source
- Vertical Sync Interrupt Source
- DMA Channel: DMA channel to transfer pixel data from camera to frame buffer
- DMA Channel Trigger Source
- Bits Per Pixel: Bits per pixel to define the size of frame line

The DRV_CAMERA_OVM7690_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handler returned by the Initialize Interface would be used by the other interfaces such as DRV_CAMERA_OVM7690_Deinitialize.

Client Access

For the application to start using an instance of the module, it must call the DRV_CAMERA_OVM7690_Open function. The DRV_CAMERA_OVM7690_Open function provides a driver handle to the OVM7690 Camera Driver instance for operations. If the driver is deinitialized using the function DRV_CAMERA_OVM7690_Deinitialize function, the application must call the DRV_CAMERA_OVM7690_Open function again to set up the instance of the driver.

Client Operations

Client operations provide the API interface for control command and pixel data transfer from the OVM7690 Camera Driver to the Graphics Frame Buffer.

Configuring the Library

Macros

	Name	Description
	DRV_OVM7690_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.

Description

The configuration of the OVM7690 Camera Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the OVM7690 Camera Driver build. Based on the selections made here and the system setup, the OVM7690 Camera Driver may support the selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

Control Commands

The following OVM7690-specific control commands are provided:

- DRV_CAMERA_OVM7690_FrameBufferAddressSet
- DRV_CAMERA_OVM7690_Start
- DRV_CAMERA_OVM7690_Stop
- DRV_CAMERA_OVM7690_FrameRectSet

Application Process

An application needs to perform following steps:

1. The system should have completed necessary setup initializations.
2. The I2C driver object should have been initialized by calling DRV_I2C_Initialize.
3. The Timer driver object should have been initialized by calling DRV_Timer_Initialize,
4. The Output Control driver object should have been initialized by calling DRV_OC_Initialize,
5. The OVM7690 Camera Driver object should have been initialized by calling DRV_CAMERA_OVM7690_Initialize,
6. Open the OVM7690 Camera Driver client by calling DRV_CAMERA_OVM7690_Open.
7. Pass the Graphics Frame buffer address to OVM7690 Camera Driver by calling DRV_CAMERA_OVM7690_FrameBufferAddressSet.
8. Set the Frame Rectangle area by calling DRV_CAMERA_OVM7690_FrameRectSet.
9. Set Other Camera settings such as: soft reset, enabling pclk, enabling href, enabling vsync, output color format, reversing HREF polarity, gating clock to the HREF, pixel clock frequency, sub-sampling mode by calling DRV_CAMERA_OVM7690_RegisterSet.
10. Start the OVM7690 Camera by calling DRV_CAMERA_OVM7690_Start.

DRV_OVM7690_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ovm7690_config_template.h

C

```
#define DRV_OVM7690_INTERRUPT_MODE false
```

Description

OVM7690 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of OVM7690 operation is desired
- false - Select if polling mode of OVM7690 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

Building the Library

This section lists the files that are available in the OVM7690 Camera Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/camera/ovm7690`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_camera_ovm7690.h	This file provides the interface definitions of the OVM7690 Camera Driver.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/drv_camera_ovm7690.c	This file contains the implementation of the OVM7690 Camera Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The OVM7690 Camera Driver Library depends on the following modules:

- I2C Driver Library
- Output Compare Driver Library
- Timer Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_CAMERA_OVM7690_Initialize	Initializes the OVM7690 Camera instance for the specified driver index.
	DRV_CAMERA_OVM7690_Deinitialize	Deinitializes the specified instance of the OVM7690 Camera Driver module.
	DRV_CAMERA_OVM7690_RegisterSet	Sets the camera OVM7690 configuration registers.
	DRV_CAMERA_OVM7690_Tasks	Maintains the OVM7690 state machine.

b) Client Setup Functions

	Name	Description
	DRV_CAMERA_OVM7690_Open	Opens the specified OVM7690 Camera Driver instance and returns a handle to it.
	DRV_CAMERA_OVM7690_Close	Closes an opened instance of the OVM7690 Camera Driver.

c) Camera-specific Functions

	Name	Description
	DRV_CAMERA_OVM7690_FrameBufferAddressSet	Sets the framebuffer address.
	DRV_CAMERA_OVM7690_FrameRectSet	Sets the frame rectangle set.
	DRV_CAMERA_OVM7690_Start	Starts camera rendering to the display.
	DRV_CAMERA_OVM7690_Stop	Stops rendering the camera Pixel data.

d) Other Functions

	Name	Description
	DRV_CAMERA_OVM7690_HsyncEventHandler	Horizontal synchronization event handler.
	DRV_CAMERA_OVM7690_VsyncEventHandler	Vertical synchronization event handler .

	_DRV_CAMERA_OVM7690_DMAEventHandler	This is function _DRV_CAMERA_OVM7690_DMAEventHandler.
	_DRV_CAMERA_OVM7690_delayMS	This is function _DRV_CAMERA_OVM7690_delayMS.
	_DRV_CAMERA_OVM7690_HardwareSetup	This is function _DRV_CAMERA_OVM7690_HardwareSetup.

e) Data Types and Constants

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_OBJ	OVM7690 Camera Driver client object.
	DRV_CAMERA_OVM7690_CLIENT_STATUS	Identifies OVM7690 Camera possible client status.
	DRV_CAMERA_OVM7690_ERROR	Identifies OVM7690 Camera possible errors.
	DRV_CAMERA_OVM7690_INIT	OVM7690 Camera Driver initialization parameters.
	DRV_CAMERA_OVM7690_OBJ	OVM7690 Camera Driver instance object.
	DRV_CAMERA_OVM7690_RECT	OVM7690 Camera window rectangle coordinates.
	DRV_CAMERA_OVM7690_REG12_OP_FORMAT	Lists OVM7690 Camera device register addresses.
	DRV_CAMERA_OVM7690_INDEX_0	OVM7690 driver index definitions.
	DRV_CAMERA_OVM7690_INDEX_1	This is macro DRV_CAMERA_OVM7690_INDEX_1.
	DRV_CAMERA_OVM7690_REG12_SOFT_RESET	OVM7690 Camera Driver Register 0x12 Soft reset flag.
	DRV_CAMERA_OVM7690_SCCB_READ_ID	OVM7690 Camera SCCB Interface device Read Slave ID.
	DRV_CAMERA_OVM7690_SCCB_WRITE_ID	OVM7690 Camera SCCB Interface device Write Slave ID.

Description

This section describes the Application Programming Interface (API) functions of the Camera Driver Library.

a) System Functions

DRV_CAMERA_OVM7690_Initialize Function

Initializes the OVM7690 Camera instance for the specified driver index.

File

drv_camera_ovm7690.h

C

```
SYS_MODULE_OBJ DRV_CAMERA_OVM7690_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the OVM7690 Camera Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the OVM7690 Camera module ID. Refer to the description of the DRV_CAMERA_OVM7690_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This function must be called before any other OVM7690 Camera Driver function is called.

This function should only be called once during system initialization unless DRV_CAMERA_OVM7690_Deinitialize is called to deinitialize the driver instance. This function will NEVER block for hardware access.

Preconditions

None.

Example

```
// The following code snippet shows an example OVM7690 driver initialization.

DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
```

```
cameraInit.hsyncInterruptSource    = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource    = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel              = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource        = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                     = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_CAMERA_OVM7690_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

)

DRV_CAMERA_OVM7690_Deinitialize Function

Deinitializes the specified instance of the OVM7690 Camera Driver module.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the OVM7690 Camera Driver module, disabling its operation (and any hardware), and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_CAMERA_OVM7690_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object; //  Returned from DRV_CAMERA_OVM7690_Initialize
SYS_STATUS          status;

DRV_CAMERA_OVM7690_Deinitialize(object);

status = DRV_CAMERA_OVM7690_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_Deinitialize(SYS_MODULE_OBJ object)

DRV_CAMERA_OVM7690_RegisterSet Function

Sets the camera OVM7690 configuration registers.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_RegisterSet(DRV_CAMERA_OVM7690_REGISTER_ADDRESS regIndex,
uint8_t regValue);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function sets the OVM7690 Camera configuration registers using the SCCB interface.

Remarks

This function can be used separately or within an interface.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

The SCCB interface also must have been initialized to configure the OVM7690 Camera Driver.

Example

```
DRV_HANDLE handle;
uint8_t reg12 = DRV_CAMERA_OVM7690_REG12_SOFT_RESET;

 handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
 if (DRV_HANDLE_INVALID == handle)
 {
     //error
     return;
 }

 if ( DRV_CAMERA_OVM7690_RegisterSet( DRV_CAMERA_OVM7690_REG12_REG_ADDR,
                                      reg12 ) !=
                                      DRV_CAMERA_OVM7690_ERROR_NONE )
 {
     //error
     return;
 }
```

Parameters

Parameters	Description
regIndex	Defines the OVM7690 configuration register addresses.
regValue	Defines the register value to be set.

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_RegisterSet

(

DRV_CAMERA_OVM7690_REGISTER_ADDRESS regIndex,

uint8_t regValue
)

DRV_CAMERA_OVM7690_Tasks Function

Maintains the OVM7690 state machine.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_CAMERA_OVM7690_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_CAMERA_OVM7690_Open Function

Opens the specified OVM7690 Camera Driver instance and returns a handle to it.

File

drv_camera_ovm7690.h

C

```
DRV_HANDLE DRV_CAMERA_OVM7690_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur:

- if the number of client objects allocated via DRV_CAMERA_OVM7690_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client
- if the driver is not ready to be opened, typically when the initialize function has not completed execution

Description

This function opens the specified OVM7690 Camera Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

Remarks

The handle returned is valid until the DRV_CAMERA_OVM7690_Close function is called. This function will NEVER block waiting for hardware.If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_CAMERA_OVM7690_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_CAMERA_OVM7690_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
)

DRV_CAMERA_OVM7690_Close Function

Closes an opened instance of the OVM7690 Camera Driver.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of the OVM7690 Camera Driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines (with one possible exception described in the "Remarks" section). A new handle must be obtained by calling DRV_CAMERA_OVM7690_Open before the caller may use the driver again

Remarks

Usually there is no need for the client to verify that the Close operation has completed. The driver will abort any ongoing operations when this function is called.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USART_Open
DRV_CAMERA_OVM7690_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

void DRV_CAMERA_OVM7690_Close(DRV_Handle handle)

c) Camera-specific Functions

DRV_CAMERA_OVM7690_FrameBufferAddressSet Function

Sets the framebuffer address.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameBufferAddressSet(DRV_HANDLE handle, void * frameBuffer);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function will set the framebuffer address. This framebuffer address will point to the location at which frame data is to be rendered. This buffer is shared with the display controller to display the frame on the display.

Remarks

This function is mandatory. A valid framebuffer address must be set to display the camera data.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;
uint16_t frameBuffer[DISP_VER_RESOLUTION][DISP_HOR_RESOLUTION];

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameBufferAddressSet( handle, (void *) frameBuffer ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameBufferAddressSet

(

DRV_HANDLE handle,

void * frameBuffer

)

DRV_CAMERA_OVM7690_FrameRectSet Function

Sets the frame rectangle set.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameRectSet(DRV_HANDLE handle, uint32_t left, uint32_t top,
uint32_t right, uint32_t bottom);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function sets the frame rectangle coordinates. The frame within the rectangle is copied to the framebuffer. The left and top values are expected to be less than right and bottom respectively. Left, top, right, and bottom values are also expected to be within range of screen coordinates. Internally it calls the DRV_CAMERA_OVM7690_RegisterSet function to set the respective registers. The rectangle coordinates are also maintained in the driver object.

Remarks

This function is optional if default values are expected to be used.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

The SCCB interface also must have been initialized to configure the OVM7690 Camera Driver.

Example

```
DRV_HANDLE handle;
uint32_t left   = 0x69;
uint32_t top    = 0x0E;
uint32_t right  = DISP_HOR_RESOLUTION + 0x69;
uint32_t bottom = DISP_VER_RESOLUTION + 0x69;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameRectSet( handle, left, top, right, bottom ) !=
                                      DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function
left	left frame coordinate
top	top frame coordinate
right	right frame coordinate
bottom	bottom frame coordinate

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameRectSet

(

DRV_HANDLE handle,

uint32_t left,

uint32_t top,

uint32_t right,

uint32_t bottom

)

DRV_CAMERA_OVM7690_Start Function

Starts camera rendering to the display.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Start(DRV_HANDLE handle);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function starts the camera rendering to the display by writing the pixel data to the framebuffer. The framebuffer is shared between the OVM7690 Camera and the display controller.

Remarks

This function is mandatory. Camera module will not update the framebuffer without calling this function.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

DRV_CAMERA_OVM7690_FrameBufferAddressSet must have been called to set a valid framebuffer address.

Example

```
DRV_HANDLE handle;
uint16_t frameBuffer[DISP_VER_RESOLUTION][DISP_HOR_RESOLUTION];

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameBufferAddressSet( handle, (void *) frameBuffer ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_Start( handle ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Start
(
    DRV_HANDLE handle
);
```

DRV_CAMERA_OVM7690_Stop Function

Stops rendering the camera Pixel data.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Stop(DRV_HANDLE handle);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function starts the camera rendering to the display by writing the pixel data to the framebuffer. The framebuffer is shared between the OVM7690 Camera and the display controller.

Remarks

This function only disables the interrupt for HSYNC and VSYNC. To stop the camera the power-down pin needs to be toggled to an active-high value., which will stop the camera internal clock and maintain the register values.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_Stop( handle ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function.

Function

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Stop
(
    DRV_HANDLE handle
);
```

d) Other Functions

DRV_CAMERA_OVM7690_HsyncEventHandler Function

Horizontal synchronization event handler.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_HsyncEventHandler(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the OVM7690 Camera sends a Horizontal Sync Pulse on the HSYNC line. It sets the next line address in the DMA module.

Remarks

This function is mandatory.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
cameraInit.hsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel               = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource         = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                      = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

void __ISR( HSYNC_ISR_VECTOR) _Ovm7690HSyncHandler(void)
{
    DRV_CAMERA_OVM7690_HsyncEventHandler(objectHandle);

    SYS_INT_SourceStatusClear(HSYNC_INTERRUPT_SOURCE);
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_HsyncEventHandler(SYS_MODULE_OBJ object)

DRV_CAMERA_OVM7690_VsyncEventHandler Function

Vertical synchronization event handler .

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_VsyncEventHandler(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the OVM7690 Camera sends a Vertical Sync Pulse on the VSYNC line. It clears the number of lines drawn variable.

Remarks

This function is mandatory.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
cameraInit.hsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel               = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource         = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                      = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

void __ISR( VSYNC_ISR_VECTOR) _Ovm7690VSyncHandler(void)
{
    DRV_CAMERA_OVM7690_VsyncEventHandler(objectHandle);

    SYS_INT_SourceStatusClear(VSYNC_INTERRUPT_SOURCE);
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_VsyncEventHandler(SYS_MODULE_OBJ object)

_DRV_CAMERA_OVM7690_DMAEventHandler Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_DMAEventHandler(SYS_DMA_TRANSFER_EVENT event, SYS_DMA_CHANNEL_HANDLE handle,
uintptr_t contextHandle);
```

Description

This is function _DRV_CAMERA_OVM7690_DMAEventHandler.

_DRV_CAMERA_OVM7690_delayMS Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_delayMS(unsigned int delayMs);
```

Description

This is function _DRV_CAMERA_OVM7690_delayMS.

_DRV_CAMERA_OVM7690_HardwareSetup Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_HardwareSetup(DRV_CAMERA_OVM7690_OBJ * dObj);
```

Description

This is function _DRV_CAMERA_OVM7690_HardwareSetup.

e) Data Types and Constants

DRV_CAMERA_OVM7690_CLIENT_OBJ Structure

OVM7690 Camera Driver client object.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  DRV_CAMERA_OVM7690_OBJ * hDriver;
  DRV_IO_INTENT ioIntent;
  bool inUse;
  DRV_CAMERA_OVM7690_ERROR error;
  DRV_CAMERA_OVM7690_CLIENT_STATUS status;
} DRV_CAMERA_OVM7690_CLIENT_OBJ;
```

Members

Members	Description
DRV_CAMERA_OVM7690_OBJ * hDriver;	The hardware instance object associated with the client
DRV_IO_INTENT ioIntent;	The I/O intent with which the client was opened
bool inUse;	This flags indicates if the object is in use or is available
DRV_CAMERA_OVM7690_ERROR error;	Driver Error
DRV_CAMERA_OVM7690_CLIENT_STATUS status;	Client status

Description

OVM7690 Camera Driver Client Object.

This structure provides a definition of the OVM7690 Camera Driver client object.

Remarks

These values are been updated into the DRV_CAMERA_OVM7690_Open function.

DRV_CAMERA_OVM7690_CLIENT_STATUS Enumeration

Identifies OVM7690 Camera possible client status.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY
} DRV_CAMERA_OVM7690_CLIENT_STATUS;
```

Members

Members	Description
DRV_CAMERA_OVM7690_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	An error has occurred.
DRV_CAMERA_OVM7690_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_CAMERA_OVM7690_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	The driver is currently busy and cannot start additional operations.
DRV_CAMERA_OVM7690_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY	The module is running and ready for additional operations

Description

OVM7690 Camera Client Status.

This enumeration defines possible OVM7690 Camera Client Status.

Remarks

This enumeration values are set by driver interfaces: DRV_CAMERA_OVM7690_Open and DRV_CAMERA_OVM7690_Close.

DRV_CAMERA_OVM7690_ERROR Enumeration

Identifies OVM7690 Camera possible errors.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE,
  DRV_CAMERA_OVM7690_ERROR_NONE
} DRV_CAMERA_OVM7690_ERROR;
```

Members

Members	Description
DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE	OVM7690 Camera Driver Invalid Handle
DRV_CAMERA_OVM7690_ERROR_NONE	OVM7690 Camera Driver error none

Description

OVM7690 Camera Error flag

This enumeration defines possible OVM7690 Camera errors.

Remarks

This enumeration values are returned by driver interfaces in case of errors.

DRV_CAMERA_OVM7690_INIT Structure

OVM7690 Camera Driver initialization parameters.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  CAMERA_MODULE_ID cameraID;
  void * sourcePort;
  PORTS_CHANNEL hsyncChannel;
  PORTS_BIT_POS hsyncPosition;
  PORTS_CHANNEL vsyncChannel;
  PORTS_BIT_POS vsyncPosition;
  INT_SOURCE hsyncInterruptSource;
  INT_SOURCE vsyncInterruptSource;
  DMA_CHANNEL dmaChannel;
  DMA_TRIGGER_SOURCE dmaTriggerSource;
  uint16_t bpp;
} DRV_CAMERA_OVM7690_INIT;
```

Members

Members	Description
CAMERA_MODULE_ID cameraID;	Camera module ID
void * sourcePort;	Source Port Address
PORTS_CHANNEL hsyncChannel;	HSYNC pin channel
PORTS_BIT_POS hsyncPosition;	HSYNC pin bit position
PORTS_CHANNEL vsyncChannel;	VSYNC pin channel
PORTS_BIT_POS vsyncPosition;	VSYNC pin bit position
INT_SOURCE hsyncInterruptSource;	HSYNC Interrupt Source
INT_SOURCE vsyncInterruptSource;	VSYNC Interrupt Source
DMA_CHANNEL dmaChannel;	DMA channel
DMA_TRIGGER_SOURCE dmaTriggerSource;	DMA trigger source
uint16_t bpp;	Bits per pixel

Description

OVM7690 Camera Initialization parameters

This structure defines OVM7690 Camera Driver initialization parameters.

Remarks

These values should be passed into the DRV_CAMERA_OVM7690_Initialize function.

DRV_CAMERA_OVM7690_OBJ Structure

OVM7690 Camera Driver instance object.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  CAMERA_MODULE_ID moduleId;
  SYS_STATUS status;
  bool inUse;
  bool isExclusive;
  size_t nClients;
  PORTS_CHANNEL hsyncChannel;
  PORTS_BIT_POS hsyncPosition;
  PORTS_CHANNEL vsyncChannel;
  PORTS_BIT_POS vsyncPosition;
  INT_SOURCE hsyncInterruptSource;
  INT_SOURCE vsyncInterruptSource;
  SYS_DMA_CHANNEL_HANDLE dmaHandle;
  DMA_CHANNEL dmaChannel;
  DMA_TRIGGER_SOURCE dmaTriggerSource;
  bool dmaTransferComplete;
  void * sourcePort;
  uint32_t frameLineCount;
```

```
  uint32_t frameLineSize;
  void * frameLineAddress;
  void * frameBufferAddress;
  DRV_CAMERA_OVM7690_RECT rect;
  uint16_t bpp;
} DRV_CAMERA_OVM7690_OBJ;
```

Members

Members	Description
CAMERA_MODULE_ID moduleId;	The module index associated with the object
SYS_STATUS status;	The status of the driver
bool inUse;	Flag to indicate this object is in use
bool isExclusive;	Flag to indicate that driver has been opened exclusively.
size_t nClients;	Keeps track of the number of clients • that have opened this driver
PORTS_CHANNEL hsyncChannel;	HSYNC pin channel
PORTS_BIT_POS hsyncPosition;	HSYNC pin bit position
PORTS_CHANNEL vsyncChannel;	VSYNC pin channel
PORTS_BIT_POS vsyncPosition;	VSYNC pin bit position
INT_SOURCE hsyncInterruptSource;	HSYNC Interrupt Source
INT_SOURCE vsyncInterruptSource;	VSYNC Interrupt Source
SYS_DMA_CHANNEL_HANDLE dmaHandle;	DMA Handle
DMA_CHANNEL dmaChannel;	Read DMA channel
DMA_TRIGGER_SOURCE dmaTriggerSource;	DMA Trigger Source
bool dmaTransferComplete;	DMA Transfer Complete Flag
void * sourcePort;	Source Port Address
uint32_t frameLineCount;	Frame Line Count
uint32_t frameLineSize;	Frame Line Size
void * frameLineAddress;	Frame Line Address
void * frameBufferAddress;	Framebuffer Address
DRV_CAMERA_OVM7690_RECT rect;	Window Rectangle
uint16_t bpp;	Bits per pixel supported

Description

OVM7690 Camera Driver Instance Object

This structure provides a definition of the OVM7690 Camera Driver instance object.

Remarks

These values are been updated into the DRV_CAMERA_OVM7690_Initialize function.

DRV_CAMERA_OVM7690_RECT Structure

OVM7690 Camera window rectangle coordinates.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  uint32_t left;
  uint32_t top;
  uint32_t right;
  uint32_t bottom;
} DRV_CAMERA_OVM7690_RECT;
```

Members

Members	Description
uint32_t left;	OVM7690 Camera Window left coordinate
uint32_t top;	OVM7690 Camera Window top coordinate
uint32_t right;	OVM7690 Camera Window right coordinate

uint32_t bottom;	OVM7690 Camera Window bottom coordinate

Description

OVM7690 Camera Window Rect

This structure defines window rectangle co-ordinates as left, right, top, and bottom.

Remarks

These values should be passed into the DRV_CAMERA_OVM7690_FrameRectSet function.

DRV_CAMERA_OVM7690_REG12_OP_FORMAT Enumeration

Lists OVM7690 Camera device register addresses.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_REG12_OP_FORMAT_RAW_2
} DRV_CAMERA_OVM7690_REG12_OP_FORMAT;
```

Members

Members	Description
DRV_CAMERA_OVM7690_REG12_OP_FORMAT_RAW_2	Bayer Raw Format

Description

OVM7690 Camera Device Register Addresses.

This enumeration defines the list of device register addresses.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_RegisterSet function. Refer to the specific device data sheet for more information.

DRV_CAMERA_OVM7690_INDEX_0 Macro

OVM7690 driver index definitions.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_INDEX_0 0
```

Description

OVM7690 Camera Driver Module Index

These constants provide OVM7690 Camera Driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_Initialize and DRV_CAMERA_OVM7690_Open routines to identify the driver instance in use.

DRV_CAMERA_OVM7690_INDEX_1 Macro

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_INDEX_1 1
```

Description

This is macro DRV_CAMERA_OVM7690_INDEX_1.

DRV_CAMERA_OVM7690_REG12_SOFT_RESET Macro

OVM7690 Camera Driver Register 0x12 Soft reset flag.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_REG12_SOFT_RESET
```

Description

OVM7690 Camera Driver Soft reset flag.

This macro provides a definition of the OVM7690 Camera Register 0x12 Soft reset flag.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CAMERA_OVM7690_SCCB_READ_ID Macro

OVM7690 Camera SCCB Interface device Read Slave ID.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_SCCB_READ_ID
```

Description

OVM7690 Camera Driver SCCB Read ID

This macro provides a definition of the OVM7690 Camera SCCB Interface device Read Slave ID.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CAMERA_OVM7690_SCCB_WRITE_ID Macro

OVM7690 Camera SCCB Interface device Write Slave ID.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_SCCB_WRITE_ID
```

Description

OVM7690 Camera Driver SCCB Write ID

This macro provides a definition of the OVM7690 Camera SCCB Interface device Write Slave ID.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_RegisterSet function to identify the OVM7690 Camera SCCB Interface device Write Slave ID.

Files

Files

Name	Description
drv_camera_ovm7690.h	OVM7690 Camera Driver local data structures.
drv_ovm7690_config_template.h	OVM7690 Device Driver configuration template.

Description

drv_camera_ovm7690.h

OVM7690 Camera Driver local data structures.

Enumerations

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_STATUS	Identifies OVM7690 Camera possible client status.
	DRV_CAMERA_OVM7690_ERROR	Identifies OVM7690 Camera possible errors.
	DRV_CAMERA_OVM7690_REG12_OP_FORMAT	Lists OVM7690 Camera device register addresses.

Functions

	Name	Description
	_DRV_CAMERA_OVM7690_delayMS	This is function _DRV_CAMERA_OVM7690_delayMS.
	_DRV_CAMERA_OVM7690_DMAEventHandler	This is function _DRV_CAMERA_OVM7690_DMAEventHandler.
	_DRV_CAMERA_OVM7690_HardwareSetup	This is function _DRV_CAMERA_OVM7690_HardwareSetup.
	DRV_CAMERA_OVM7690_Close	Closes an opened instance of the OVM7690 Camera Driver.
	DRV_CAMERA_OVM7690_Deinitialize	Deinitializes the specified instance of the OVM7690 Camera Driver module.
	DRV_CAMERA_OVM7690_FrameBufferAddressSet	Sets the framebuffer address.
	DRV_CAMERA_OVM7690_FrameRectSet	Sets the frame rectangle set.
	DRV_CAMERA_OVM7690_HsyncEventHandler	Horizontal synchronization event handler.
	DRV_CAMERA_OVM7690_Initialize	Initializes the OVM7690 Camera instance for the specified driver index.
	DRV_CAMERA_OVM7690_Open	Opens the specified OVM7690 Camera Driver instance and returns a handle to it.
	DRV_CAMERA_OVM7690_RegisterSet	Sets the camera OVM7690 configuration registers.
	DRV_CAMERA_OVM7690_Start	Starts camera rendering to the display.
	DRV_CAMERA_OVM7690_Stop	Stops rendering the camera Pixel data.
	DRV_CAMERA_OVM7690_Tasks	Maintains the OVM7690 state machine.
	DRV_CAMERA_OVM7690_VsyncEventHandler	Vertical synchronization event handler .

Macros

	Name	Description
	DRV_CAMERA_OVM7690_INDEX_0	OVM7690 driver index definitions.
	DRV_CAMERA_OVM7690_INDEX_1	This is macro DRV_CAMERA_OVM7690_INDEX_1.
	DRV_CAMERA_OVM7690_REG12_SOFT_RESET	OVM7690 Camera Driver Register 0x12 Soft reset flag.
	DRV_CAMERA_OVM7690_SCCB_READ_ID	OVM7690 Camera SCCB Interface device Read Slave ID.
	DRV_CAMERA_OVM7690_SCCB_WRITE_ID	OVM7690 Camera SCCB Interface device Write Slave ID.

Structures

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_OBJ	OVM7690 Camera Driver client object.
	DRV_CAMERA_OVM7690_INIT	OVM7690 Camera Driver initialization parameters.
	DRV_CAMERA_OVM7690_OBJ	OVM7690 Camera Driver instance object.
	DRV_CAMERA_OVM7690_RECT	OVM7690 Camera window rectangle coordinates.

Description

OVM7690 Camera Driver Local Data Structures

This header file provides the local data structures for the OVM7690 Camera Driver Library.

File Name

drv_camera_ovm7690.h

Company

Microchip Technology Inc.

drv_ovm7690_config_template.h

OVM7690 Device Driver configuration template.

Macros

	Name	Description
	DRV_OVM7690_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.

Description

OVM7690 Device Driver Configuration Template

This header file contains the build-time configuration selections for the OVM7690 device driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_ovm7690_config_template.h

Company

Microchip Technology Inc.

CAN Driver Library

This section describes the CAN Driver Library.

Introduction

The CAN Static Driver provides a high-level interface to manage the CAN module on the Microchip family of microcontrollers.

Description

Through MHC, this driver provides an API to initialize the CAN module, as well as the baud rate. The API also allows simple transmit and receive functionality.

Library Interface

Function(s)

	Name	Description
	DRV_CAN_ChannelMessageReceive	Receives a message on a channel for the specified driver index. **Implementation:** Static
	DRV_CAN_ChannelMessageTransmit	Transmits a message on a channel for the specified driver index. **Implementation:** Static
	DRV_CAN_Close	Closes the CAN instance for the specified driver index. **Implementation:** Static
	DRV_CAN_Deinitialize	Deinitializes the DRV_CAN_Initialize instance that has been called for the specified driver index. **Implementation:** Static
	DRV_CAN_Initialize	Initializes the CAN instance for the specified driver index. **Implementation:** Static
	DRV_CAN_Open	Opens the CAN instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the CAN Driver Library.

Function(s)

DRV_CAN_ChannelMessageReceive Function

Receives a message on a channel for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
bool DRV_CAN_ChannelMessageReceive(CAN_CHANNEL channelNum, int address, uint8_t DLC, uint8_t* message);
```

Returns

- true - When a message has been received
- false - When a message has not been received

Description

This routine receives data into a buffer from the CAN bus according to the channel, address, and data length given.

Remarks

This routine receives a standard or extended messages based upon the CAN Driver setup.

Preconditions

DRV_CAN_Initialize has been called.

Parameters

Parameters	Description
CAN_CHANNEL channelNum	CAN channel to use
int address	CAN address to receive on
uint8_t DLC	Data Length Code of Message
uint8_t* message	Pointer to put the message data to receive

Function

bool DRV_CAN_ChannelMessageReceive(CAN_CHANNEL channelNum, int address,
uint8_t DLC, uint8_t* message);

DRV_CAN_ChannelMessageTransmit Function

Transmits a message on a channel for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
bool DRV_CAN_ChannelMessageTransmit(CAN_CHANNEL channelNum, int address, uint8_t DLC, uint8_t* message);
```

Returns

Boolean "true" when a message has been transmitted.

Description

This routine transmits a data buffer on the CAN bus according to the channel, address, and data length given.

Remarks

This routine receives a standard or extended messages based upon the CAN Driver setup.

Preconditions

DRV_CAN_Initialize has been called.

Parameters

Parameters	Description
CAN_CHANNEL channelNum	CAN channel to use
int address	CAN address to transmit on
uint8_t DLC	Data Length Code of Message
uint8_t* message	Pointer to the message data to send

Function

bool DRV_CAN_ChannelMessageTransmit(CAN_CHANNEL channelNum, int address,
uint8_t DLC, uint8_t* message);

DRV_CAN_Close Function

Closes the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Close();
```

Returns

None.

Description

This routine closes the CAN driver instance for the specified driver instance, making it ready for clients to use it.

Preconditions

DRV_CAN_Initialize has been called.

Function

void DRV_CAN_Close(void)

DRV_CAN_Deinitialize Function

Deinitializes the DRV_CAN_Initialize instance that has been called for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Deinitialize();
```

Returns

None.

Description

This routine deinitializes the CAN Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Preconditions

None.

Function

void DRV_CAN_Deinitialize(void)

DRV_CAN_Initialize Function

Initializes the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Initialize();
```

Returns

None.

Description

This routine initializes the CAN Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This routine must be called before any other CAN routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_CAN_Initialize(void)

DRV_CAN_Open Function

Opens the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Open();
```

Returns

None.

Description

This routine opens the CAN Driver instance for the specified driver instance, making it ready for clients to use it.

Preconditions

DRV_CAN_Initialize has been called.

Function

void DRV_CAN_Open(void)

Codec Driver Libraries

This section describes the Codec Driver Libraries available in MPLAB Harmony.

AK4384 Codec Driver Library

This topic describes the AK4384 Codec Driver Library.

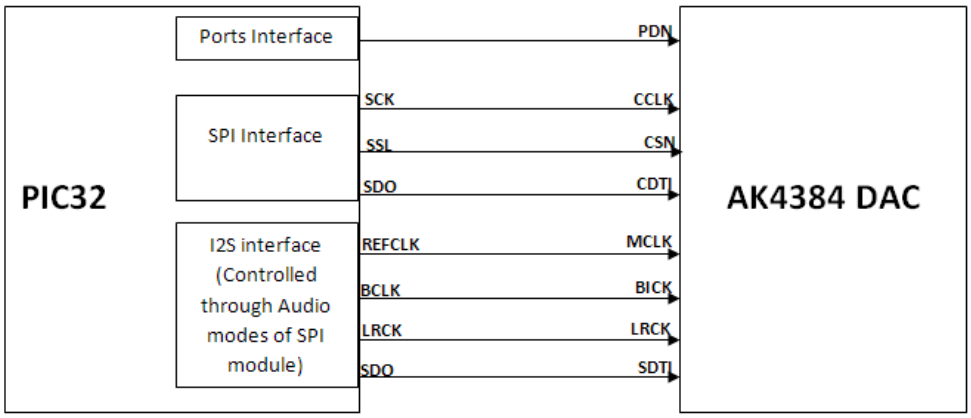
Introduction

This library provides an interface to manage the AK4384 106 dB 192 kHz 24-Bit DAC that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4384 module is 24-bit Audio DAC from Asahi Kasei Microdevices Corporation. The AK4384 can be interfaced to Microchip microcontrollers through SPI and I2S serial interfaces. SPI interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4384 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4384 Codec Driver supports the following features:

- Sampling Rate Ranging from 8 kHz to 192 kHz
- 128 times Oversampling (Normal Speed mode)
- 64 times Oversampling (Double Speed mode)
- 32 times Oversampling (Quad Speed mode)
- Digital de-emphasis for 32k, 44.1k and 48 kHz sampling
- Soft mute
- Digital Attenuator (Linear 256 steps)
- I/F format:
- 24-bit MSB justified
- 24/20/16-bit LSB justified
- I2S
- Master clock:
- 256 fs, 384 fs, 512 fs, 768 fs, or 1152 fs (Normal Speed mode)
- 128 fs, 192 fs, 256 fs, or 384 fs (Double Speed mode)
- 128 fs or 192 fs (Quad Speed mode)

Using the Library

This topic describes the basic architecture of the AK4384 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ak4384.h`

The interface to the AK4384 Codec Driver library is defined in the `drv_ak4384.h` header file. Any C language source (`.c`) file that uses the AK4384 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

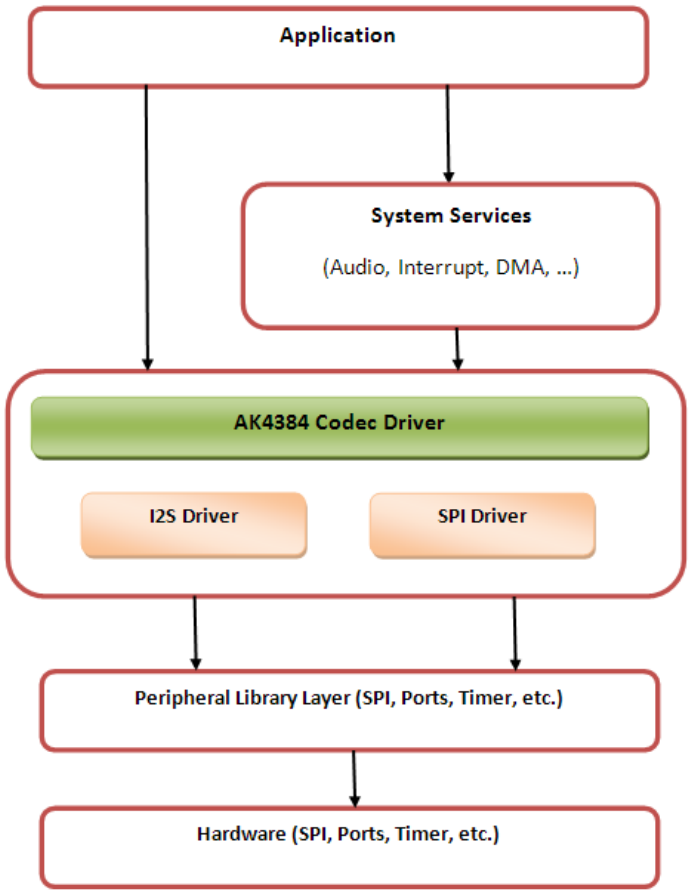
Abstraction Model

This library provides a low-level abstraction of the AK4384 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4384 Codec Driver is positioned in the MPLAB Harmony framework. The AK4384 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4384 module.

AK4384 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4384 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4384 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4384 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Codec Specific Functions	Provides functions that are Codec-specific.
Data Transfer Functions	Provides data transfer functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4384 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic provides information on system initialization, implementations, and provides a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4384 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4384_INIT or by using Initialization Overrides) that are supported by the specific AK4384 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- SPI driver module index. The module index should be same as the one used in initializing the SPI Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Master clock detection mode
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK4384_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4384_Deinitialize, DRV_ AK4384_Status and DRV_I2S_Tasks.

Implementations

The AK4384 Codec Driver can have the following implementations:

Implementation	Description	MPLAB Harmony Components
Implementation 1	Dedicated hardware for control (SPI) and data (I2S) interface.	Standard MPLAB Harmony drivers for SPI and I2S interfaces.
Implementation 2	Dedicated hardware for data (I2S) interface. Ports pins for control interface.	Standard MPLAB Harmony drivers for I2S interface. Virtual MPLAB Harmony drivers for SPI interface.
Implementation 3	Dedicated hardware for data (I2S) interface. Ports pins for control.	Standard MPLAB Harmony drivers for I2S interface. An internal bit-banged implementation of control interface in the AK4384 Codec Driver.

If Implementation 3 is in use, while initializing fields of DRV_AK4384_INIT structure, the SPI Driver module index initialization is redundant. The user can pass a dummy value.

For Implementation 3, the user has to additionally initialize parameters to support bit-banged control interface implementation. These additional parameters can be passed by assigning values to the respective macros in `system_config.h`.

Example:

```
DRV_AK4384_INIT drvak4384Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .volume = 120,
    .mclkMode = DRV_AK4384_MCLK_MODE_MANUAL,
    .queueSizeTransmit = 2,
};

/*
 The SPI module index should be same as the one used in
 initializing the SPI driver.
 The SPI module index initialization is redundant
 if Implementation 3 is in use.
 */
drvak4384Init.spiDriverModuleIndex = DRV_SPI_INDEX_0;

/*
 The I2S module index should be same as the one used in
 initializing the I2S driver.
 */
drvak4384Init.i2sDriverModuleIndex = DRV_I2S_INDEX_0;

ak4384DevObject = DRV_AK4384_Initialize(DRV_AK4384_INDEX_0, (SYS_MODULE_INIT *) &drvak4384Init);
if (SYS_MODULE_OBJ_INVALID == ak4384DevObject)
{
```

```
    // Handle error
}
```

Task Routine

The DRV_AK4384_Tasks will be called from the System Task Service.

Client Access

This topic describes client access and includes a code example.

Description

For the application to start using an instance of the module, it must call the DRV_AK4384_Open function. The DRV_AK4384_Open provides a driver handle to the AK4384 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4384_Deinitialize, the application must call the DRV_AK4384_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the AK4384 Codec Driver can be known by calling DRV_AK4384_Status.

Example:

```
DRV_HANDLE handle;
SYS_STATUS ak4384Status;
    ak4384Status = DRV_AK4384_Status(sysObjects.ak4384DevObject);
        if (SYS_STATUS_READY == ak4384Status)
        {
                // The driver can now be opened.
                appData.ak4384Client.handle = DRV_AK4384_Open
                                              (DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                if(appData.ak4384Client.handle != DRV_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_AK4384_SET_BUFFER_HANDLER;
                }
                else
                {
                    SYS_DEBUG(0, "Find out what's wrong \r\n");
                }
    }
    else
    {
        /* AK4384 Driver Is not ready */
                ;
    }
```

Client Operations

This topic describes client operations and provides a code example.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4384 Codec.

The following AK4384 Codec specific control command functions are provided:

Notes:

1. The calling and execution of the following functions does not guarantee that the function (and its associated Codec command) has been set in the Codec peer interfaced through the SPI. It just means that the submission of the command has started over the SPI.
2. Regarding Note 1, the user should not call the following functions consecutively, which could result in unexpected behavior. If needed, the user should confirm the completion status of a function before calling any of the other functions.
3. To know the completion status of the following functions, users can register a command event callback handler by calling the function 'DRV_AK4384_CommandEventHandlerSet'. The callback handler will be called when the last submitted command (submitted by calling one of the following functions) has completed.

- DRV_AK4384_SamplingRateSet
- DRV_AK4384_SamplingRateGet
- DRV_AK4384_VolumeSet
- DRV_AK4384_VolumeGet
- DRV_AK4384_MuteOn

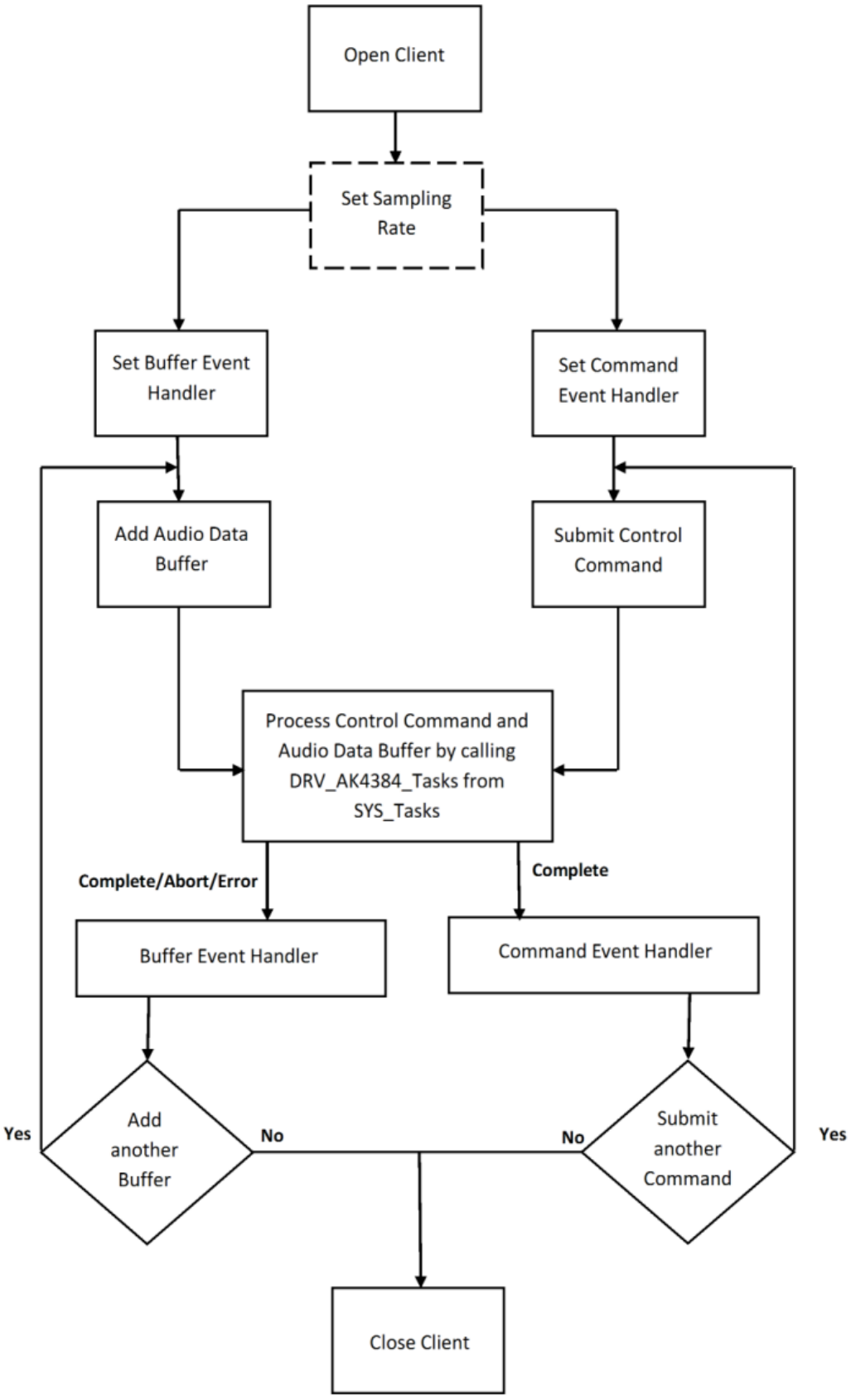
- DRV_AK4384_MuteOff
- DRV_AK4384_ZeroDetectEnable
- DRV_AK4384_ZeroDetectDisable
- DRV_AK4384_ZeroDetectModeSet
- DRV_AK4384_ZeroDetectInvertEnable
- DRV_AK4384_ZeroDetectInvertDisable
- DRV_AK4384_ChannelOutputInvertEnable
- DRV_AK4384_ChannelOutputInvertDisable
- DRV_AK4384_SlowRollOffFilterEnable
- DRV_AK4384_SlowRollOffFilterDisable
- DRV_AK4384_DeEmphasisFilterSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK4384 Codec. A notification for the submitted requests can be received by registering a command callback event with the driver. The driver notifies by calling the callback on successfully transmitting the command to the AK4384 Codec module.

The function DRV_AK4384_BufferAddWrite is a buffered data operation functions. This function schedules non-blocking audio data transfer operation. The function adds the request to the hardware instance queues and returns a buffer handle. The requesting client also registers a callback event with the driver. The driver notifies the client with DRV_AK4384_BUFFER_EVENT_COMPLETE, DRV_AK4384_BUFFER_EVENT_ERROR, or DRV_AK4384_BUFFER_EVENT_ABORT events.

The submitted control commands and audio buffer add requests are processed under DRV_AK4384_Tasks function. This function is called from the SYS_Tasks routine.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.

2. The I2S Driver object should have been initialized by calling DRV_I2S_Initialize.
3. The SPI Driver object should have been initialized by calling DRV_SPI_Initialize.
4. The AK4384 Codec Driver object should be initialized by calling DRV_AK4384_Initialize.
5. The necessary sampling rate value should be set up by calling DRV_AK4384_ SamplingRateSet.
6. Register buffer event handler for the client handle by calling DRV_AK4384_BufferEventHandlerSet.
7. Register command event handler for the client handle by calling DRV_AK4384_CommandEventHandlerSet.
8. Submit a command by calling specific command API.
9. Add a buffer to initiate the data transfer by calling DRV_AK4384_BufferAddWrite.
10. The submitted command and Audio data processing happens b calling DRV_AK4384_Tasks from SYS_Tasks.
11. Repeat steps 9 through 10 to handle multiple buffer transmission and reception.
12. When the client is done, it can use DRV_AK4384_Close to close the client handle.

Example:

```
typedef enum
{
    APP_STATE_AK4384_OPEN,
    APP_STATE_AK4384_SET_COMMAND_HANDLER,
    APP_STATE_AK4384_SET_BUFFER_HANDLER,
    APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND,
    APP_STATE_AK4384_ADD_BUFFER,
    APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE,
    APP_STATE_AK4384_BUFFER_COMPLETE
} APP_STATES;

typedef struct
{
    DRV_HANDLE handle;
    DRV_AK4384_BUFFER_HANDLE writeBufHandle;
    DRV_AK4384_BUFFER_EVENT_HANDLER bufferHandler;
    DRV_AK4384_COMMAND_EVENT_HANDLER commandHandler;
    uintptr_t context;
    uint8_t *txbufferObject;
    size_t bufferSize;

} APP_AK4384_CLIENT;

typedef struct
{
    /* Application's current state*/
    APP_STATES state;
    /* USART client handle */
    APP_AK4384_CLIENT ak4384Client;
} APP_DATA;
APP_DATA appData;
SYS_MODULE_OBJ ak4384DevObject;
DRV_AK4384_INIT drvak4384Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .volume = 120,
    .mclkMode = DRV_AK4384_MCLK_MODE_MANUAL,
    .queueSizeTransmit = 2,
};

void SYS_Initialize(void * data)
{
    /*
     The SPI module index should be same as the one used in
     initializing the SPI driver.
     The SPI module index initialization is redundant
     if Implementation 3 (Described in System Access) is in use.
     */
    drvak4384Init.spiDriverModuleIndex = DRV_SPI_INDEX_0;

    /*
     The I2S module index should be same as the one used in
     initializing the I2S driver.
     */
    drvak4384Init.i2sDriverModuleIndex = DRV_I2S_INDEX_0;
```

```
    ak4384DevObject = DRV_AK4384_Initialize(DRV_AK4384_INDEX_0, (SYS_MODULE_INIT *) & drvak4384Init);
    if (SYS_MODULE_OBJ_INVALID == ak4384DevObject) {
        // Handle error
    }
}

void APP_Tasks (void )
{
    switch(appData.state)
    {
        /* Open the ak4384 client and get an Handle */
        case APP_STATE_AK4384_OPEN:
        {
            SYS_STATUS ak4384Status;
            ak4384Status = DRV_AK4384_Status(sysObjects.ak4384DevObject);
            if (SYS_STATUS_READY == ak4384Status)
            {
                // This means the driver can now be opened.
                appData.ak4384Client.handle = DRV_AK4384_Open(DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                if(appData.ak4384Client.handle != DRV_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_AK4384_SET_COMMAND_HANDLER;
                }
                else
                {
                    SYS_DEBUG(0, "Find out what is wrong \r\n");
                }
            }
            else
            {
                /* Wait for AK4384 to Initialize */
                ;
            }
        }
        break;

        /* Register a command event handler */
        case APP_STATE_AK4384_SET_COMMAND_HANDLER:
        {
            DRV_AK4384_CommandEventHandlerSet(appData.ak4384Client.handle,
                    appData.ak4384Client.commandHandler,
                    appData.ak4384Client.context);
            appData.state = APP_STATE_AK4384_SET_BUFFER_HANDLER;
        }
        break;


        /* Register a buffer event handler */
        case APP_STATE_AK4384_SET_BUFFER_HANDLER:
        {
            DRV_AK4384_BufferEventHandlerSet(appData.ak4384Client.handle,
                    appData.ak4384Client.bufferHandler,
                    appData.ak4384Client.context);
            appData.state = APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND;
        }
        break;

        /* Submit a set sampling rate command */
        case APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND:
        {
            DRV_AK4384_SamplingRateSet(appData.ak4384Client.handle,48000);
            appData.state = APP_STATE_AK4384_ADD_BUFFER;
        }
        break;

        /* Add the Audio buffer to be transmitted */
        case APP_STATE_AK4384_ADD_BUFFER:
        {
```

```
            DRV_AK4384_BufferAddWrite(appData.ak4384Client.handle, &appData.ak4384Client.writeBufHandle,
            appData.ak4384Client.txbufferObject, appData.ak4384Client.bufferSize);
            if(appData.ak4384Client.writeBufHandle != DRV_AK4384_BUFFER_HANDLE_INVALID)
            {
                appData.state = APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE;
            }
            else
            {
                SYS_DEBUG(0, "Find out what is wrong \r\n");
            }
        }
        break;

    /* Audio Buffer transmission under process */
        case APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE:
        {
        }
        break;

    /* Audio Buffer transmission completed */
        case APP_STATE_AK4384_BUFFER_COMPLETE:
        {
            /* Add another buffer */
            appData.state = APP_STATE_AK4384_ADD_BUFFER;
        }
        break;

        default:
        {
        }
        break;
    }

}

void APP_AK4384CommandEventHandler(uintptr_t context )
{

    // Last submitted command successful. Take action as needed.
}

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE handle, uintptr_t context )
{

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:
        {
            // Can set appData.state = APP_STATE_AK4384_BUFFER_COMPLETE;
        // Take Action as needed

        }
        break;
        case DRV_AK4384_BUFFER_EVENT_ERROR:
        {
        // Take Action as needed

        } break;

        case DRV_AK4384_BUFFER_EVENT_ABORT:
        {
        // Take Action as needed

        } break;

    }
}
```

```
void SYS_Tasks(void)
{
    DRV_AK4384_Tasks(ak4384DevObject);
    APP_Tasks();
}
```

Configuring the Library

Macros

	Name	Description
	DRV_AK4384_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4384_CONTROL_CLOCK	Sets up clock frequency for the control interface (SPI)
	DRV_AK4384_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4384_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4384_TIMER_DRIVER_MODULE_INDEX	Identifies the Timer Module Index for custom virtual SPI driver implementation.
	DRV_AK4384_TIMER_PERIOD	Identifies the period for the bit bang timer.
	DRV_AK4384_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency
	DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency

Description

The configuration of the AK4384 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4384 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4384 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4384_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_CLIENTS_NUMBER DRV_AK4384_INSTANCES_NUMBER
```

Description

AK4384 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4384 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4384_CONTROL_CLOCK Macro

Sets up clock frequency for the control interface (SPI)

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_CONTROL_CLOCK
```

Description

AK4384 Control Interface Clock Speed configuration

Sets up clock frequency for the control interface (SPI). The maximum value supported is 5MHZ.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_INPUT_REFCLOCK
```

Description

AK4384 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4384_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_INSTANCES_NUMBER
```

Description

AK4384 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4384 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4384_TIMER_DRIVER_MODULE_INDEX Macro

Identifies the Timer Module Index for custom virtual SPI driver implementation.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_TIMER_DRIVER_MODULE_INDEX
```

Description

AK4384 Timer Module Index

Identifies the Timer Module Index for custom virtual SPI driver implementation. The AK4384 uses SPI protocol for control interface. The Timer Module Index is needed by AK4384 driver to implement a virtual SPI driver for control command exchange with the AK4384 CODEC.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_TIMER_PERIOD Macro

Identifies the period for the bit bang timer.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_TIMER_PERIOD
```

Description

AK4384 Timer Period

Identifies the period for the bit bang timer after which the timer interrupt should occur. The value assigned should align with the expected control interface clock defined by AK4384_CONTROL_CLOCK.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_BCLK_BIT_CLK_DIVISOR
```

Description

AK4384 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K I2S sampling frequency

Following BCLK to LRCK ratios are supported 16bit LSB Justified >=32fs 20bit LSB Justified >=40fs 24bit MSB Justified >=48fs 24bit I2S Compatible >=48fs 24bit LSB Justified >=48fs

Typical values for the divisor are 1,2,4 and 8

Remarks

None.

DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4384 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K I2S sampling frequency

Supported MCLK to LRCK Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs [Normal Speed Mode(8kHz~48kHz)] 128fs, 192fs, 256fs or 384fs [Double Speed Mode(60kHz~96kHz)] 128fs, 192fs [Quad Speed Mode(120kHz~192kHz)]

Remarks

None

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4384 Codec Driver, I2S Driver, and the Timer Driver.

Figure 1: AK4384 Codec Driver MHC Configuration

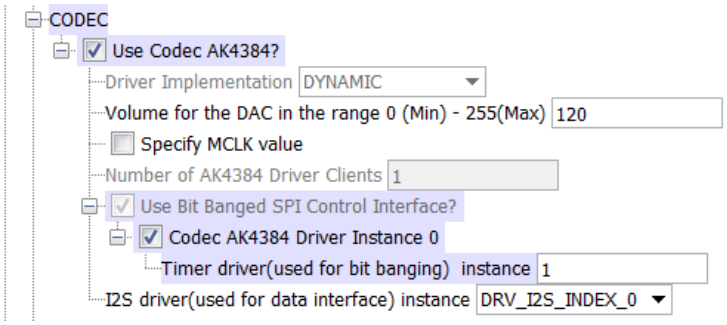


Figure 2: I2S Driver MHC Configuration

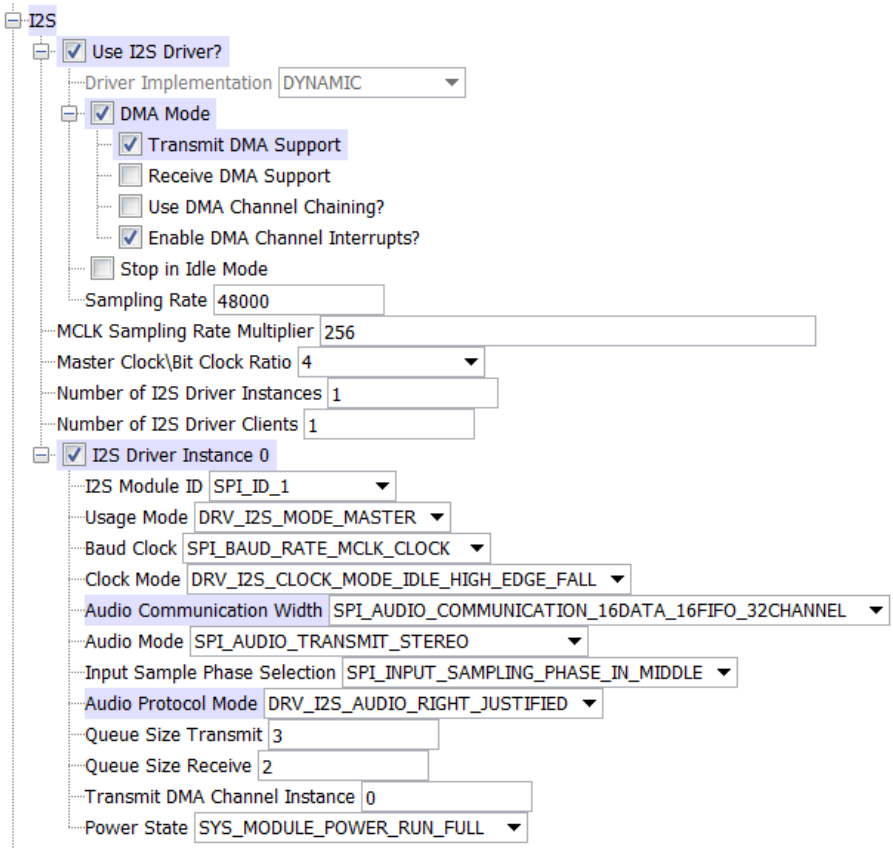
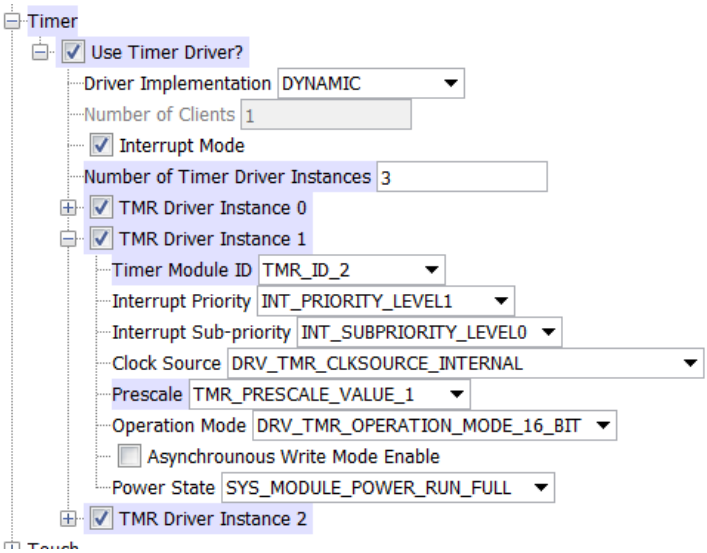


Figure 3: Timer Driver MHC Configuration



Building the Library

This section lists the files that are available in the AK4384 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4384 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4384`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak4384.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak4384_bit_banged_control_interface.c`	This file contains implementation of the AK4384 Codec Driver with a custom bit-banged implementation for control interface driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_ak4384_virtual_control_interface.c`	This file contains implementation of the AK4384 Codec Driver with a virtual SPI driver as control interface driver. **Note:** This file is currently unsupported.
`/src/dynamic/drv_ak4384.c`	This file contains the core implementation of the AK4384 Codec Driver **Note:** This file currently unsupported.

Module Dependencies

The AK4384 Driver Library depends on the following modules:

- I2S Driver Library
- SPI Driver Library

- Timer Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4384_Initialize	Initializes hardware and data for the instance of the AK4384 DAC module. **Implementation:** Dynamic
	DRV_AK4384_Deinitialize	Deinitializes the specified instance of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Status	Gets the current status of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4384_SetAudioCommunicationMode	This function provides a run time audio format configuration

b) Client Setup Functions

	Name	Description
	DRV_AK4384_Open	Opens the specified AK4384 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4384_Close	Closes an opened-instance of the AK4384 driver. **Implementation:** Dynamic

c) Codec Specific Functions

	Name	Description
	DRV_AK4384_ChannelOutputInvertDisable	Disables output polarity of the selected Channel. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertEnable	Enables output polarity of the selected channel. **Implementation:** Dynamic
	DRV_AK4384_DeEmphasisFilterSet	Allows specifies enabling of digital de-emphasis filter. **Implementation:** Dynamic
	DRV_AK4384_MuteOff	Disables AK4384 output for soft mute. **Implementation:** Dynamic
	DRV_AK4384_MuteOn	Allows AK4384 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateGet	This function gets the sampling rate set on the DAC AK4384. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterDisable	Disables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterEnable	Enables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_VolumeGet	This function gets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_VolumeSet	This function sets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectDisable	Disables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectEnable	Enables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertDisable	Disables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertEnable	Enables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectModeSet	Sets mode of AK4384 channel-independent zeros detect function. **Implementation:** Dynamic

d) Data Transfer Functions

	Name	Description
	DRV_AK4384_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4384_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_AK4384_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_AK4384_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_AK4384_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic

e) Other Functions

	Name	Description
	DRV_AK4384_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4384_VersionGet	Returns the version of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_VersionStrGet	Returns the version of AK4384 driver in string format. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_AK4384_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4384_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4384_BUFFER_EVENT_HANDLER	Pointer to a AK4384 Driver Buffer Event handler function.
	DRV_AK4384_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4384_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4384_COMMAND_EVENT_HANDLER	Pointer to a AK4384 Driver Command Event Handler Function
	DRV_AK4384_DEEMPHASIS_FILTER	Identifies de-emphasis filter function.
	DRV_AK4384_INIT	Defines the data required to initialize or reinitialize the AK4384 driver.
	DRV_AK4384_MCLK_MODE	Identifies the mode of master clock to AK4384 DAC.
	DRV_AK4384_ZERO_DETECT_MODE	Identifies Zero Detect Function mode
	DRV_AK4384_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4384_COUNT	Number of valid AK4384 driver indices.
	DRV_AK4384_INDEX_0	AK4384 driver index definitions.
	DRV_AK4384_INDEX_1	This is macro DRV_AK4384_INDEX_1.
	DRV_AK4384_INDEX_2	This is macro DRV_AK4384_INDEX_2.
	DRV_AK4384_INDEX_3	This is macro DRV_AK4384_INDEX_3.
	DRV_AK4384_INDEX_4	This is macro DRV_AK4384_INDEX_4.
	DRV_AK4384_INDEX_5	This is macro DRV_AK4384_INDEX_5.

Description

This section describes the API functions of the AK4384 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4384_Initialize Function

Initializes hardware and data for the instance of the AK4384 DAC module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
SYS_MODULE_OBJ DRV_AK4384_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4384 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4384 routine is called.

This routine should only be called once during system initialization unless DRV_AK4384_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. DRV_SPI_Initialize must be called if SPI driver is used for handling the control interface of this CODEC driver.

Example

```
DRV_AK4384_INIT                init;
SYS_MODULE_OBJ                 objectHandle;

init.moduleInit.value          = SYS_MODULE_POWER_RUN_FULL;
init.spiDriverModuleIndex      = DRV_SPI_INDEX_0;  // This will be ignored for a custom
                                                   // control interface driver implementation
init.i2sDriverModuleIndex      = DRV_I2S_INDEX_0;
init.mclkMode                  = DRV_AK4384_MCLK_MODE_MANUAL;
init.audioDataFormat           = DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S;
init.powerDownPortChannel      = PORT_CHANNEL_G;
init.powerDownBitPosition      = PORTS_BIT_POS_15;

objectHandle = DRV_AK4384_Initialize(DRV_AK4384_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK4384_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT *const init

);

DRV_AK4384_Deinitialize Function

Deinitializes the specified instance of the AK4384 driver module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4384 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4384_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4384_Initialize
SYS_STATUS          status;


DRV_AK4384_Deinitialize(object);

status = DRV_AK4384_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4384_Initialize routine

Function

void DRV_AK4384_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4384_Status Function

Gets the current status of the AK4384 driver module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
SYS_STATUS DRV_AK4384_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4384 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4384_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4384_Initialize
SYS_STATUS          ak4384Status;

ak4384Status = DRV_AK4384_Status(object);
if (SYS_STATUS_READY == ak4384Status)
{
    // This means the driver can be opened using the
    // DRV_AK4384_Open function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4384_Initialize routine

Function

SYS_STATUS DRV_AK4384_Status(SYS_MODULE_OBJ object)

DRV_AK4384_Tasks Function

Maintains the driver's control and data interface state machine.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4384_Initialize

while (true)
{
    DRV_AK4384_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4384_Initialize)

Function

void DRV_AK4384_Tasks(SYS_MODULE_OBJ object);

DRV_AK4384_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_ak4384.h

C

```
void DRV_AK4384_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_AK4384_SetAudioCommunicationMode
(
DRV_HANDLE handle,
const DATA_LENGTH dl,
const SAMPLE_LENGTH sl
)

b) Client Setup Functions

DRV_AK4384_Open Function

Opens the specified AK4384 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ak4384.h

C

```
DRV_HANDLE DRV_AK4384_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under following conditions:

- if the number of client objects allocated via DRV_AK4384_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the ioIntent options passed are not relevant to this driver

Description

This routine opens the specified AK4384 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

Only DRV_IO_INTENT_WRITE is a valid ioIntent option as AK4384 is DAC only.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4384_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4384_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4384_Open(DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4384_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4384_Close Function

Closes an opened-instance of the AK4384 driver.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the AK4384 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4384_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4384_Open

DRV_AK4384_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_Close(DRV_Handle handle)

c) Codec Specific Functions

DRV_AK4384_ChannelOutputInvertDisable Function

Disables output polarity of the selected Channel.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ChannelOutputInvertDisable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This function disables output polarity of the selected Channel.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ChannelOutputInvertDisable(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Left or Right channel

Function

void DRV_AK4384_ChannelOutputInvertDisable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_ChannelOutputInvertEnable Function

Enables output polarity of the selected channel.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ChannelOutputInvertEnable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This function enables output polarity of the selected channel.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ChannelOutputInvertEnable(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Left or Right channel

Function

void DRV_AK4384_ChannelOutputInvertEnable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_DeEmphasisFilterSet Function

Allows specifies enabling of digital de-emphasis filter.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_DeEmphasisFilterSet(DRV_HANDLE handle, DRV_AK4384_DEEMPHASIS_FILTER filter);
```

Returns

None.

Description

This function allows specifies enabling of digital de-emphasis for 32, 44.1 or 48 kHz sampling rates (tc = 50/15 µs)

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_DeEmphasisFilterSet(myAK4384Handle, DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ)
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
filter	Specifies Enable of de-emphasis filter

Function

void DRV_AK4384_DeEmphasisFilterSet

(

DRV_HANDLE handle,

DRV_AK4384_DEEMPHASIS_FILTER filter

)

DRV_AK4384_MuteOff Function

Disables AK4384 output for soft mute.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4384 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

    DRV_AK4384_MuteOff(myAK4384Handle); //AK4384 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_MuteOff(DRV_HANDLE handle)

DRV_AK4384_MuteOn Function

Allows AK4384 output for soft mute on.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4384 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_MuteOn(myAK4384Handle);  //AK4384 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_MuteOn(DRV_HANDLE handle);

DRV_AK4384_SamplingRateGet Function

This function gets the sampling rate set on the DAC AK4384.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint32_t DRV_AK4384_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

This function gets the sampling rate set on the DAC AK4384.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t baudRate;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

baudRate = DRV_AK4384_SamplingRateGet(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4384_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4384_SamplingRateSet Function

This function sets the sampling rate of the media stream.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SamplingRateSet(myAK4384Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
baudRate	Baud Rate to be set

Function

void DRV_AK4384_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4384_SlowRollOffFilterDisable Function

Disables Slow Roll-off filter function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SlowRollOffFilterDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables Slow Roll-off filter function. Sharp Roll-off filter function gets enabled.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SlowRollOffFilterDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_SlowRollOffFilterDisable(DRV_HANDLE handle);

DRV_AK4384_SlowRollOffFilterEnable Function

Enables Slow Roll-off filter function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SlowRollOffFilterEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables Slow Roll-off filter function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SlowRollOffFilterEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_SlowRollOffFilterEnable(DRV_HANDLE handle);

DRV_AK4384_VolumeGet Function

This function gets the volume for AK4384 Codec.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint8_t DRV_AK4384_VolumeGet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This functions gets the current volume programmed to the DAC AK4384.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

  volume = DRV_AK4384_VolumeGet(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT_RIGHT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to get.

Function

uint8_t DRV_AK4384_VolumeGet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_VolumeSet Function

This function sets the volume for AK4384 Codec.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_VolumeSet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan, uint8_t volume);
```

Returns

None.

Description

This functions sets the volume value from 0-255, which can attenuate from 0 dB to –48 dB and mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

    DRV_AK4384_VolumeSet(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT_RIGHT, 120); //Step 120 volume
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set
volume	volume value from 0-255, which can attenuate from 0 dB to –48 dB and mute

Function

void DRV_AK4384_VolumeSet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan, uint8_t volume)

DRV_AK4384_ZeroDetectDisable Function

Disables AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4384 channel-independent zeros detect function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectDisable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectEnable Function

Enables AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables AK4384 channel-independent zeros detect function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectEnable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectInvertDisable Function

Disables inversion of polarity for zero detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectInvertDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables inversion of polarity for zero detect function. DZF goes “H” at Zero Detection.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectInvertDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectInvertDisable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectInvertEnable Function

Enables inversion of polarity for zero detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectInvertEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables inversion of polarity for zero detect function. DZF goes “L” at Zero Detection

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectInvertEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectInvertEnable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectModeSet Function

Sets mode of AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectModeSet(DRV_HANDLE handle, DRV_AK4384_ZERO_DETECT_MODE zdMode);
```

Returns

None.

Description

This function sets mode of AK4384 channel-independent zeros detect function

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectModeSet(myAK4384Handle, DRV_AK4384_ZERO_DETECT_MODE_ANDED);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
zdMode	Specifies zero detect function mode.

Function

void DRV_AK4384_ZeroDetectModeSet
(
DRV_HANDLE handle,
DRV_AK4384_ZERO_DETECT_MODE zdMode
)

d) Data Transfer Functions

DRV_AK4384_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4384_BUFFER_HANDLE * bufferHandle, void * buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4384_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4384_BUFFER_HANDLE_INVALID if:

- a buffer could not be allocated to the request
- the input buffer pointer is NULL
- the buffer size is '0'
- the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4384_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4384_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4384 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4384 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 device instance and the DRV_AK4384_Status must have returned SYS_STATUS_READY.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4384_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle,
                APP_AK4384BufferEventHandler, (uintptr_t)&myAppObj);
```

```
DRV_AK4384_BufferAddWrite(myAK4384handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4384 instance as return by the DRV_AK4384_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4384_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK4384_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4384_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4384_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4384_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle,
                APP_AK4384BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddWrite(myAK4384handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4384_BufferEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK4384_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK4384_BufferCombinedQueueSizeGet Function

This function returns the number of bytes queued (to be processed) in the buffer queue.

Implementation: Dynamic

File

drv_ak4384.h

C

```
size_t DRV_AK4384_BufferCombinedQueueSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired client handle.

Description

This function returns the number of bytes queued (to be processed) in the buffer queue associated with the driver instance to which the calling client belongs. The client can use this function to know number of bytes that is in the queue to be transmitted.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead/DRV_AK4384_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                 (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                 myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
```

```
// The user can get to know dynamically available data in the queue to be
// transmitted by calling DRV_AK4384_BufferCombinedQueueSizeGet
bufferQueuedSize = DRV_AK4384_BufferCombinedQueueSizeGet(myAK4384Handle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

size_t DRV_AK4384_BufferCombinedQueueSizeGet(DRV_HANDLE handle)

DRV_AK4384_BufferQueueFlush Function

This function flushes off the buffers associated with the client object.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferQueueFlush(const DRV_HANDLE handle);
```

Returns

None.

Description

This function flushes off the buffers associated with the client object and disables the DMA channel used for transmission.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead/DRV_AK4384_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can stop the data processing and flushoff the data
// in the queue by calling DRV_AK4384_BufferQueueFlush
DRV_AK4384_BufferQueueFlush(myAK4384Handle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

void DRV_AK4384_BufferQueueFlush(DRV_HANDLE handle)

DRV_AK4384_BufferProcessedSizeGet Function

This function returns number of bytes that have been processed for the specified buffer.

Implementation: Dynamic

File

drv_ak4384.h

C

```
size_t DRV_AK4384_BufferProcessedSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. If this function is called on a invalid buffer handle, or if the buffer handle has expired, the function returns 0.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified I2S driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead, DRV_AK4384_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
```

```
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_AK4384_BufferProcessedSizeGet(myAK4384Handle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_AK4384_BufferProcessedSizeGet(DRV_HANDLE handle)

e) Other Functions

DRV_AK4384_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4384_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4384_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4384 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_CommandEventHandlerSet(myAK4384Handle,
                APP_AK4384CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4384_DeEmphasisFilterSet(myAK4384Handle, DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4384CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4384_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4384_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4384_VersionGet Function

Returns the version of the AK4384 driver.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint32_t DRV_AK4384_VersionGet();
```

Returns

Returns the version of AK4384 driver.

Description

The version number returned from the DRV_AK4384_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t ak4384version;
ak4384version = DRV_AK4384_VersionGet();
```

Function

uint32_t DRV_AK4384_VersionGet(void)

DRV_AK4384_VersionStrGet Function

Returns the version of AK4384 driver in string format.

Implementation: Dynamic

File

drv_ak4384.h

C

```
int8_t* DRV_AK4384_VersionStrGet();
```

Returns

returns a string containing the version of AK4384 driver.

Description

The DRV_AK4384_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4384 driver's version number. is the AK4384 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals '00'). is an optional release type ('a' for alpha, 'b' for beta not the entire word spelled out) that is not included if the release is a production version (i.e., not an alpha or beta).

The String does not contain any spaces.

Remarks

None.

Preconditions

None.

Example 1

"0.03a" "1.00"

Example 2

```
int8_t *ak4384string;
ak4384string = DRV_AK4384_VersionStrGet();
```

Function

int8_t* DRV_AK4384_VersionStrGet(void)

f) Data Types and Constants

DRV_AK4384_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_AUDIO_DATA_FORMAT_16BIT_RIGHT_JUSTIFIED = 0,
  DRV_AK4384_AUDIO_DATA_FORMAT_20BIT_RIGHT_JUSTIFIED,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_LEFT_JUSTIFIED,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_RIGHT_JUSTIFIED
} DRV_AK4384_AUDIO_DATA_FORMAT;
```

Members

Members	Description
DRV_AK4384_AUDIO_DATA_FORMAT_16BIT_RIGHT_JUSTIFIED = 0	16 bit Right Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_20BIT_RIGHT_JUSTIFIED	20 bit Right Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_LEFT_JUSTIFIED	24 bit Left Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S	24 bit I2S Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_RIGHT_JUSTIFIED	24 bit Right Justified Audio data format

Description

AK4384 Audio data format

This enumeration identifies Serial Audio data interface format.

Remarks

None.

DRV_AK4384_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_BUFFER_EVENT_COMPLETE,
  DRV_AK4384_BUFFER_EVENT_ERROR,
  DRV_AK4384_BUFFER_EVENT_ABORT
} DRV_AK4384_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4384_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4384_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4384_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4384 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4384_BufferAddWrite function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4384_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4384_BUFFER_EVENT_HANDLER Type

Pointer to a AK4384 Driver Buffer Event handler function.

File

drv_ak4384.h

C

```
typedef void (* DRV_AK4384_BUFFER_EVENT_HANDLER)(DRV_AK4384_BUFFER_EVENT event, DRV_AK4384_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4384 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4384 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4384_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4384_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4384_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4384_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4384_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4384_BUFFER_EVENT event,
                               DRV_AK4384_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4384_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4384.h

C

```
typedef uintptr_t DRV_AK4384_BUFFER_HANDLE;
```

Description

AK4384 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4384_BufferAddWrite function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_AK4384_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_CHANNEL_LEFT,
  DRV_AK4384_CHANNEL_RIGHT,
  DRV_AK4384_CHANNEL_LEFT_RIGHT,
  DRV_AK4384_NUMBER_OF_CHANNELS
} DRV_AK4384_CHANNEL;
```

Description

AK4384 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4384_COMMAND_EVENT_HANDLER Type

Pointer to a AK4384 Driver Command Event Handler Function

File

drv_ak4384.h

C

```
typedef void (* DRV_AK4384_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4384 Driver Command Event Handler Function

This data type defines the required function signature for the AK4384 driver command event handling callback function.

A command is a control instruction to the AK4384 Codec. For example, Mute ON/OFF, Zero Detect Enable/Disable, etc.

A client must register a pointer to a command event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4384_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4384CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4384_DEEMPHASIS_FILTER Enumeration

Identifies de-emphasis filter function.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ,
  DRV_AK4384_DEEMPHASIS_FILTER_OFF,
  DRV_AK4384_DEEMPHASIS_FILTER_48KHZ,
  DRV_AK4384_DEEMPHASIS_FILTER_32KHZ
} DRV_AK4384_DEEMPHASIS_FILTER;
```

Members

Members	Description
DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ	De-Emphasis filter for 44.1kHz.
DRV_AK4384_DEEMPHASIS_FILTER_OFF	De-Emphasis filter Off This is the default setting.
DRV_AK4384_DEEMPHASIS_FILTER_48KHZ	De-Emphasis filter for 48kHz.
DRV_AK4384_DEEMPHASIS_FILTER_32KHZ	De-Emphasis filter for 32kHz.

Description

AK4384 De-Emphasis Filter

This enumeration identifies the settings for de-emphasis filter function.

Remarks

None.

DRV_AK4384_INIT Structure

Defines the data required to initialize or reinitialize the AK4384 driver.

File

drv_ak4384.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  uint8_t volume;
  DRV_AK4384_MCLK_MODE mclkMode;
  bool delayDriverInitialization;
} DRV_AK4384_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies control module(SPI) driver ID for control interface of Codec
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of Codec
uint8_t volume;	Volume
DRV_AK4384_MCLK_MODE mclkMode;	Set MCLK mode.
bool delayDriverInitialization;	true if driver initialization should be delayed due to shared RESET pin

Description

AK4384 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4384 Codec driver.

Remarks

None.

DRV_AK4384_MCLK_MODE Enumeration

Identifies the mode of master clock to AK4384 DAC.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_MCLK_MODE_MANUAL,
  DRV_AK4384_MCLK_MODE_AUTO
} DRV_AK4384_MCLK_MODE;
```

Members

Members	Description
DRV_AK4384_MCLK_MODE_MANUAL	Master clock frequency mode Manual
DRV_AK4384_MCLK_MODE_AUTO	Master clock frequency mode Auto This is the default mode.

Description

AK4384 Master clock frequency mode

This enumeration identifies mode of master clock to AK4384 DAC. In Manual Setting Mode, the sampling speed is set by setting DFS0/1 bits in Control Register 2. The frequency of MCLK at each sampling speed is set automatically. In Auto Setting Mode, the MCLK frequency is detected automatically

Remarks

None.

DRV_AK4384_ZERO_DETECT_MODE Enumeration

Identifies Zero Detect Function mode

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_ZERO_DETECT_MODE_CHANNEL_SEPARATED,
  DRV_AK4384_ZERO_DETECT_MODE_ANDED
} DRV_AK4384_ZERO_DETECT_MODE;
```

Members

Members	Description
DRV_AK4384_ZERO_DETECT_MODE_CHANNEL_SEPARATED	Zero Detect channel separated. When the input data at each channel is continuously zeros for 8192 LRCK cycles, DZF pin of each channel goes to “H” This is the default mode.

DRV_AK4384_ZERO_DETECT_MODE_ANDED	Zero Detect Anded DZF pins of both channels go to “H” only when the input data at both channels are continuously zeros for 8192 LRCK cycles

Description

AK4384 Zero Detect mode

This enumeration identifies the mode of zero detect function

Remarks

None.

DRV_AK4384_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4384.h

C

```
#define DRV_AK4384_BUFFER_HANDLE_INVALID ((DRV_AK4384_BUFFER_HANDLE)(-1))
```

Description

AK4384 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4384_BufferAddWrite function if the buffer add request was not successful.

Remarks

None.

DRV_AK4384_COUNT Macro

Number of valid AK4384 driver indices.

File

drv_ak4384.h

C

```
#define DRV_AK4384_COUNT
```

Description

AK4384 Driver Module Count

This constant identifies the maximum number of AK4384 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4384 instances on this microcontroller.

Remarks

This value is device-specific.

DRV_AK4384_INDEX_0 Macro

AK4384 driver index definitions.

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_0 0
```

Description

Driver AK4384 Module Index

These constants provide AK4384 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4384_Initialize and DRV_AK4384_Open routines to identify the driver instance in use.

DRV_AK4384_INDEX_1 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_1 1
```

Description

This is macro DRV_AK4384_INDEX_1.

DRV_AK4384_INDEX_2 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_2 2
```

Description

This is macro DRV_AK4384_INDEX_2.

DRV_AK4384_INDEX_3 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_3 3
```

Description

This is macro DRV_AK4384_INDEX_3.

DRV_AK4384_INDEX_4 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_4 4
```

Description

This is macro DRV_AK4384_INDEX_4.

DRV_AK4384_INDEX_5 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_5 5
```

Description

This is macro DRV_AK4384_INDEX_5.

Files

Files

Name	Description
drv_ak4384.h	AK4384 Codec Driver Interface header file
drv_ak4384_config_template.h	AK4384 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4384Codec Driver Library.

drv_ak4384.h

AK4384 Codec Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4384_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4384_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4384_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4384_DEEMPHASIS_FILTER	Identifies de-emphasis filter function.
	DRV_AK4384_MCLK_MODE	Identifies the mode of master clock to AK4384 DAC.
	DRV_AK4384_ZERO_DETECT_MODE	Identifies Zero Detect Function mode

Functions

	Name	Description
	DRV_AK4384_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4384_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_AK4384_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_AK4384_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic
	DRV_AK4384_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertDisable	Disables output polarity of the selected Channel. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertEnable	Enables output polarity of the selected channel. **Implementation:** Dynamic
	DRV_AK4384_Close	Closes an opened-instance of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4384_DeEmphasisFilterSet	Allows specifies enabling of digital de-emphasis filter. **Implementation:** Dynamic
	DRV_AK4384_Deinitialize	Deinitializes the specified instance of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Initialize	Initializes hardware and data for the instance of the AK4384 DAC module. **Implementation:** Dynamic
	DRV_AK4384_MuteOff	Disables AK4384 output for soft mute. **Implementation:** Dynamic
	DRV_AK4384_MuteOn	Allows AK4384 output for soft mute on. **Implementation:** Dynamic

	DRV_AK4384_Open	Opens the specified AK4384 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateGet	This function gets the sampling rate set on the DAC AK4384. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4384_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK4384_SlowRollOffFilterDisable	Disables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterEnable	Enables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_Status	Gets the current status of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4384_VersionGet	Returns the version of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_VersionStrGet	Returns the version of AK4384 driver in string format. **Implementation:** Dynamic
	DRV_AK4384_VolumeGet	This function gets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_VolumeSet	This function sets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectDisable	Disables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectEnable	Enables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertDisable	Disables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertEnable	Enables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectModeSet	Sets mode of AK4384 channel-independent zeros detect function. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_AK4384_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4384_COUNT	Number of valid AK4384 driver indices.
	DRV_AK4384_INDEX_0	AK4384 driver index definitions.
	DRV_AK4384_INDEX_1	This is macro DRV_AK4384_INDEX_1.
	DRV_AK4384_INDEX_2	This is macro DRV_AK4384_INDEX_2.
	DRV_AK4384_INDEX_3	This is macro DRV_AK4384_INDEX_3.
	DRV_AK4384_INDEX_4	This is macro DRV_AK4384_INDEX_4.
	DRV_AK4384_INDEX_5	This is macro DRV_AK4384_INDEX_5.

Structures

	Name	Description
	DRV_AK4384_INIT	Defines the data required to initialize or reinitialize the AK4384 driver.

Types

	Name	Description
	DRV_AK4384_BUFFER_EVENT_HANDLER	Pointer to a AK4384 Driver Buffer Event handler function.
	DRV_AK4384_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4384_COMMAND_EVENT_HANDLER	Pointer to a AK4384 Driver Command Event Handler Function

Description

AK4384 Codec Driver Interface

The AK4384 Codec device driver interface provides a simple interface to manage the AK4384 106 dB 192 kHz 24-Bit DAC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4384 Codec device driver.

File Name

drv_ak4384.h

Company

Microchip Technology Inc.

drv_ak4384_config_template.h

AK4384 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4384_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency
	DRV_AK4384_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4384_CONTROL_CLOCK	Sets up clock frequency for the control interface (SPI)
	DRV_AK4384_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4384_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency
	DRV_AK4384_TIMER_DRIVER_MODULE_INDEX	Identifies the Timer Module Index for custom virtual SPI driver implementation.
	DRV_AK4384_TIMER_PERIOD	Identifies the period for the bit bang timer.

Description

AK4384 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4384_config_template.h

Company

Microchip Technology Inc.

AK4642 Codec Driver Library

This topic describes the AK4642 Codec Driver Library.

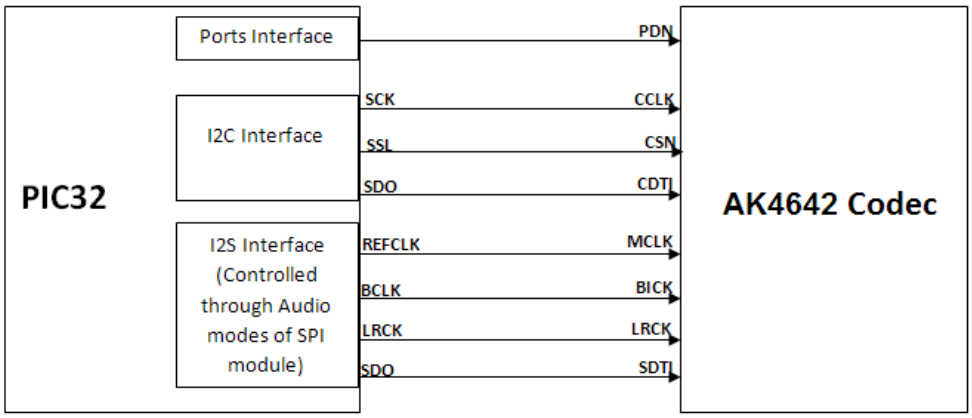
Introduction

This library provides an interface to manage the AK4642 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4642 module is 16/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK4642 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4642 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4642 Codec Driver supports the following features:

- Audio Interface Format: MSB first
- ADC: 16-bit MSB justified, 16/24-bit I2S
- DAC: 16-bit MSB justified, 16bit LSB justified, 16/24-bit I2S
- Sampling Frequency Range: 8 kHz to 48 kHz
- Digital Volume Control: +12dB ~ .115dB, 0.5dB Step
- SoftMute: On and Off
- Master Clock Frequencies: 32 fs/64 fs/128fs/256fs

Using the Library

This topic describes the basic architecture of the AK4642 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ak4642.h`

The interface to the AK4642 Codec Driver library is defined in the `drv_ak4642.h` header file. Any C language source (`.c`) file that uses the AK4642 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

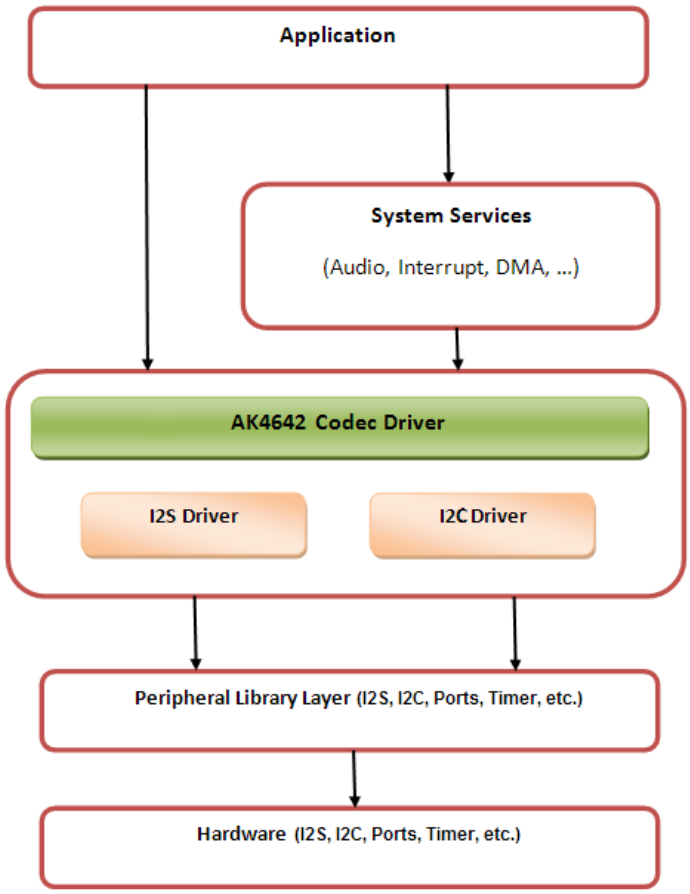
Abstraction Model

This library provides a low-level abstraction of the AK4642 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4642 Codec Driver is positioned in the MPLAB Harmony framework. The AK4642 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4642 module.

AK4642 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4642 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4642 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4642 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Codec Specific Functions	Provides functions that are codec specific.
Data Transfer Functions	Provides data transfer functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4642 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic provides information on system initialization, implementations, and provides a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4642 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4642_INIT or by using Initialization Overrides) that are supported by the specific AK4642 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Master clock detection mode
- Power down pin port initialization

The DRV_AK4642_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4642_Deinitialize, DRV_ AK4642_Status and DRV_I2S_Tasks.

Implementations

The AK4642 Codec Driver can have the following implementations:

Implementation	Description	MPLAB Harmony Components
Implementation 1	Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.
Implementation 2	Dedicated hardware for data (I2S) interface. Ports pins for control interface.	Standard MPLAB Harmony drivers for I2S interface. Virtual MPLAB Harmony drivers for I2C interface.

Example:

```
DRV_AK4642_INIT drvak4642Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4642_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4642_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4642_VOLUME,
};

/*
  The I2C and I2S module index should be same as the one used in
  initializing the I2C and I2S drivers.
 */

ak4642DevObject = DRV_AK4642_Initialize(DRV_AK4642_INDEX_0, (SYS_MODULE_INIT *) &drvak4642Init);
if (SYS_MODULE_OBJ_INVALID == ak4642DevObject)
{
    // Handle error
}
```

Task Routine

The DRV_AK4642_Tasks will be called from the System Task Service.

Client Access

This topic describes client access and includes a code example.

Description

For the application to start using an instance of the module, it must call the DRV_AK4642_Open function. The DRV_AK4642_Open provides a driver handle to the AK4642 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4642_Deinitialize, the application must call the DRV_AK4642_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the AK4642 Codec Driver can be known by calling DRV_AK4642_Status.

Example:

```
DRV_HANDLE handle;
SYS_STATUS ak4642Status;
    ak4642Status = DRV_AK4642_Status(sysObjects.ak4642DevObject);
        if (SYS_STATUS_READY == ak4642Status)
        {
               // The driver can now be opened.
               appData.ak4642Client.handle = DRV_AK4642_Open
                                            (DRV_AK4642_INDEX_0,
                                             DRV_IO_INTENT_WRITE |
                                             DRV_IO_INTENT_EXCLUSIVE );
               if(appData.ak4642Client.handle != DRV_HANDLE_INVALID)
               {
                   appData.state = APP_STATE_AK4642_SET_BUFFER_HANDLER;
               }
               else
               {
                   SYS_DEBUG(0, "Find out what's wrong \r\n");
               }
    }
    else
    {
        /* AK4642 Driver Is not ready */
                ;
    }
```

Client Operations

This topic describes client operations and provides a code example.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4642 Codec.

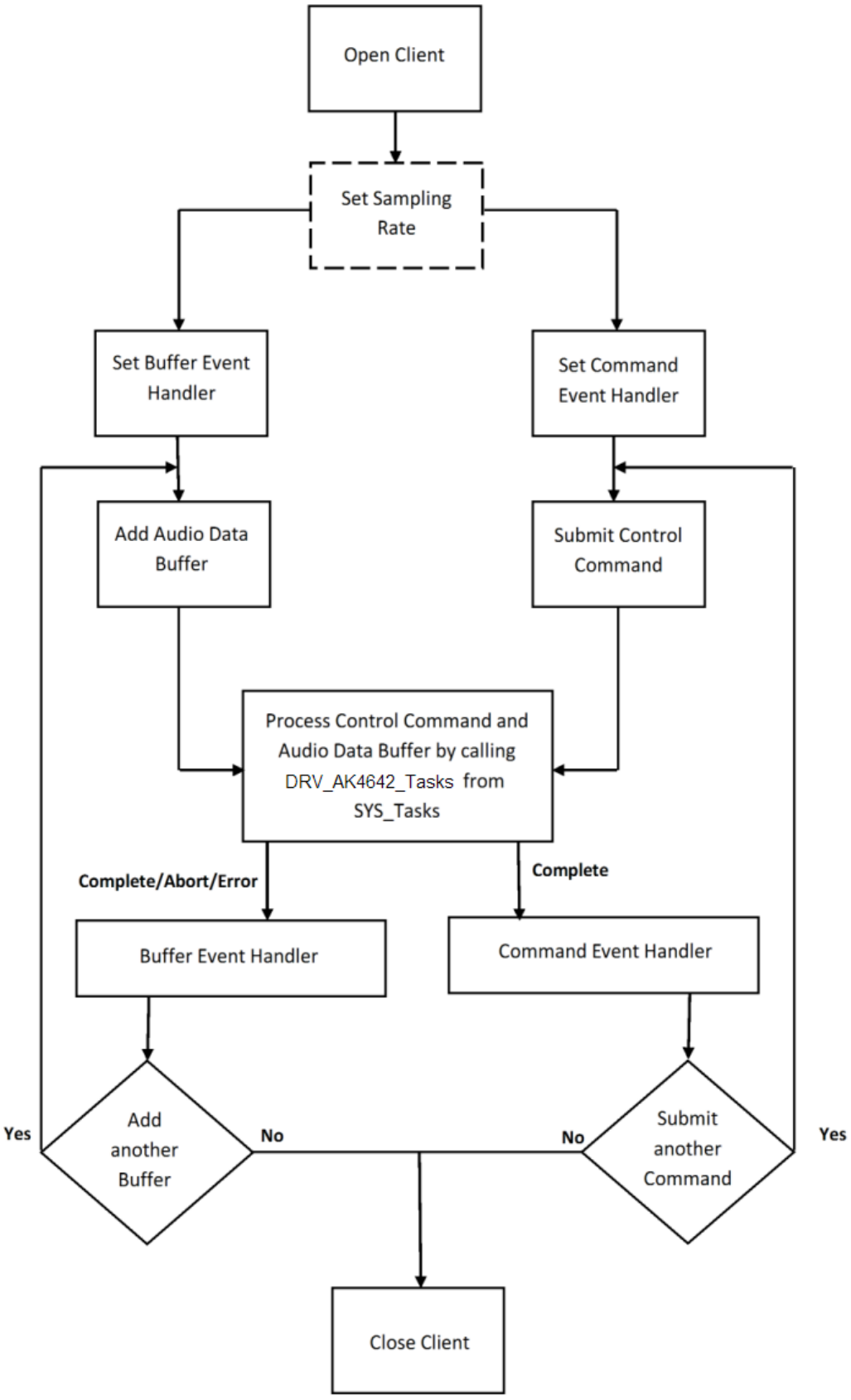
The following AK4642 Codec specific control command functions are provided:

- DRV_AK4642_SamplingRateSet
- DRV_AK4642_SamplingRateGet
- DRV_AK4642_VolumeSet
- DRV_AK4642_VolumeGet
- DRV_AK4642_MuteOn
- DRV_AK4642_MuteOff
- DRV_AK4642_IntExtMicSet
- DRV_AK4642_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the I2C Driver transmit queue, where the request is processed immediately if it is the first request, or it is processed when the previous request is complete.

DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, and DRV_AK4642_BufferAddWriteRead are buffered data operation functions. These functions schedule non-blocking audio data transfer operations. These functions add the request to the I2S Driver transmit or receive buffer queue depending on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK4642_BUFFER_EVENT_COMPLETE, DRV_AK4642_BUFFER_EVENT_ERROR, or DRV_AK4642_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.

2. The I2S driver object should have been initialized by calling DRV_I2S_Initialize.
3. The I2C driver object should have been initialized by calling DRV_I2C_Initialize.
4. The AK4642 driver object should be initialized by calling DRV_AK4642_Initialize.
5. The necessary sampling rate value should be set up by calling DRV_AK4642_ SamplingRateSet.
6. Register buffer event handler for the client handle by calling DRV_AK4642_BufferEventHandlerSet.
7. Submit a command by calling specific command API.
8. Add a buffer to initiate the data transfer by calling DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, and DRV_AK4642_BufferAddWriteRead.
9. Call the DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, or DRV_AK4642_BufferAddWriteRead function for handling multiple buffer transmissions or receptions.
10. When the client is done, it can use DRV_AK4642_Close to close the client handle.

Example:

```
typedef enum
{
    APP_STATE_AK4642_OPEN,
    APP_STATE_AK4642_SET_BUFFER_HANDLER,
    APP_STATE_AK4642_ADD_FIRST_BUFFER_READ,
    APP_STATE_AK4642_ADD_BUFFER_OUT,
    APP_STATE_AK4642_ADD_BUFFER_IN,
    APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE,
} APP_STATES;

typedef struct
{
    DRV_HANDLE handle;
    DRV_AK4642_BUFFER_HANDLE writereadBufHandle;
    DRV_AK4642_BUFFER_EVENT_HANDLER bufferEventHandler;
    uintptr_t context;
    uint8_t *txbufferObject;
    uint8_t *rxbufferObject;
    size_t bufferSize;

} APP_AK4642_CLIENT;

typedef struct
{
    /* Application's current state*/
    APP_STATES state;
    /* USART client handle */
    APP_AK4642_CLIENT ak4642Client;
} APP_DATA;
APP_DATA appData;
SYS_MODULE_OBJ ak4642DevObject;
DRV_AK4642_INIT drvak4642Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4642_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4642_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4642_VOLUME,
};

void SYS_Initialize(void * data)
{
    /* Initialize Drivers */
    DRV_I2C0_Initialize();
    sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)
                                        &drvI2S0InitData);

    sysObj.drvak4642Codec0 = DRV_AK4642_Initialize(DRV_AK4642_INDEX_0,
                             (SYS_MODULE_INIT *)&drvak4642Codec0InitData);

    /* Initialize System Services */
    SYS_INT_Initialize();
}

void APP_Tasks (void )
{
```

```
switch(appData.state)
{
    case APP_STATE_AK4642_OPEN:
    {
        SYS_STATUS status;
        status = DRV_CODEC_Status(sysObjdrvCodec0);
        if (SYS_STATUS_READY == status)
        {
        /* A client opens the driver object to get an Handle */
        appData.ak4642Client.handle = DRV_AK4642_Open(DRV_AK4642_INDEX_0,
                          DRV_IO_INTENT_WRITE|DRV_IO_INTENT_EXCLUSIVE);
        if(appData.ak4642Client.handle != DRV_HANDLE_INVALID)
        {
            appData.state = APP_STATE_AK4642_SET_BUFFER_HANDLER;
        }
        else
        {
           /* Got an Invalid Handle. Wait for AK4642 to Initialize */
        }
      }
    }
    break;

    /* Set a handler for the audio buffer completion event */
    case APP_STATE_AK4642_SET_BUFFER_HANDLER:
    {

       DRV_AK4642_BufferEventHandlerSet(appData.ak4642Client.handle,
                appData.ak4642Client.bufferEventHandler,
                appData.ak4642Client.context);

        appData.state = APP_STATE_AK4642_ADD_FIRST_BUFFER_READ;
    }
    break;

    case APP_STATE_AK4642_ADD_FIRST_BUFFER_READ:
    {
        DRV_AK4642_BufferAddWriteRead(appData.ak4642Client.handle,
                &appData.ak4642Client.writeReadBufHandle,
                appData.ak4642Client.txbufferObject,
                appData.ak4642Client.rxbufferObject,
                appData.ak4642Client.bufferSize);
        if(appData.ak4642Client.writeReadBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
        {
            appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;
        }
        else
        {
            SYS_DEBUG(0, "Find out what is wrong \r\n");
        }

    }
    break;
    /* Add an audio buffer to the ak4642 driver to be transmitted to
     * AK4642 CODEC */
    case APP_STATE_AK4642_ADD_BUFFER_OUT:
    {

        DRV_AK4642_BufferAddWrite(appData.ak4642Client.handle, &appData.ak4642Client.writeBufHandle,
        appData.ak4642Client.txbufferObject, appData.ak4642Client.bufferSize);
        if(appData.ak4642Client.writeBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
        {

            appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;
        }
        else
        {
            SYS_DEBUG(0, "Find out what is wrong \r\n");
        }
```

```
        }
        break;
        /* Add an audio buffer to the ak4642 driver to be received
         * AK4642 CODEC */
        case APP_STATE_AK4642_ADD_BUFFER_IN:
        {

            DRV_AK4642_BufferAddRead(appData.ak4642Client.handle, &appData.ak4642Client.readBufHandle,
            appData.ak4642Client.rxbufferObject, appData.ak4642Client.bufferSize);


            if(appData.ak4642Client.readBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
            {
                appData.state = APP_STATE_AK4642_ADD_BUFFER_OUT;
            }
            else
            {
                SYS_DEBUG(0, "Find out what is wrong \r\n");
            }
        }
        break;
        /* Audio data Transmission under process */
        case APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE:
        {
            /*Do nothing*/
        }
        break;

        default:
        {
        }
        break;
    }

}

/*******************************************************
 * Application AK4642 buffer Event handler.
 * This function is called back by the AK4642 driver when
 * a AK4642 data buffer RX completes.
 *******************************************************/
void APP_AK4642MicBufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE handle, uintptr_t context )
{
    static uint8_t cnt = 0;

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:
        {

            bufnum ^= 1;

            if(bufnum ==0)
            {
                appData.ak4642Client.rxbufferObject = (uint8_t *) micbuf1;
                appData.ak4642Client.txbufferObject = (uint8_t *) micbuf2;
            }
            else if(bufnum ==1)
            {
                appData.ak4642Client.rxbufferObject = (uint8_t *) micbuf2;
                appData.ak4642Client.txbufferObject = (uint8_t *) micbuf1;
            }

                DRV_AK4642_BufferAddWriteRead(appData.ak4642Client.handle,
                &appData.ak4642Client.writeReadBufHandle,
                appData.ak4642Client.txbufferObject,
                appData.ak4642Client.rxbufferObject,
                appData.ak4642Client.bufferSize);
```

```
                appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;

            }
            break;
            case DRV_AK4642_BUFFER_EVENT_ERROR:
            {
            } break;

            case DRV_AK4642_BUFFER_EVENT_ABORT:
            {
            } break;

        }
    }

    void SYS_Tasks(void)
    {
        DRV_AK4642_Tasks(ak4642DevObject);
        APP_Tasks();
    }
```

Configuring the Library

Macros

	Name	Description
	DRV_AK4642_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4642_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4642_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.

Description

The configuration of the AK4642 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4642 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4642 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4642_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_BCLK_BIT_CLK_DIVISOR
```

Description

AK4642 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

Following BCLK to LRCK ratios are supported 16bit data 16 bit channel :- 32fs, hence divisor would be 8 16bit data 32 bit channel :- 64fs, hence divisor would be 4

Remarks

None.

DRV_AK4642_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_CLIENTS_NUMBER DRV_AK4642_INSTANCES_NUMBER
```

Description

AK4642 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4642 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4642_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_INPUT_REFCLOCK
```

Description

AK4642 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4642_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_INSTANCES_NUMBER
```

Description

AK4642 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4642 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4642 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency I2S sampling frequency

Supported MCLK to Sampling frequency Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs

Remarks

None

DRV_AK4642_MCLK_SOURCE Macro

Indicate the input clock frequency to generate the MCLK to codec.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_MCLK_SOURCE
```

Description

AK4642 Data Interface Master Clock Speed configuration

Indicate the input clock frequency to generate the MCLK to codec.

Remarks

None.

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4642 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK4642 Codec Driver MHC Configuration

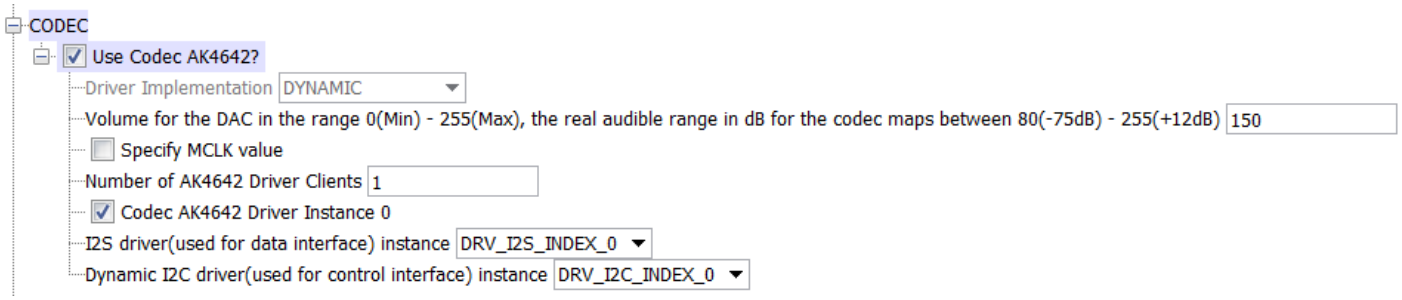


Figure 2: I2S Driver MHC Configuration

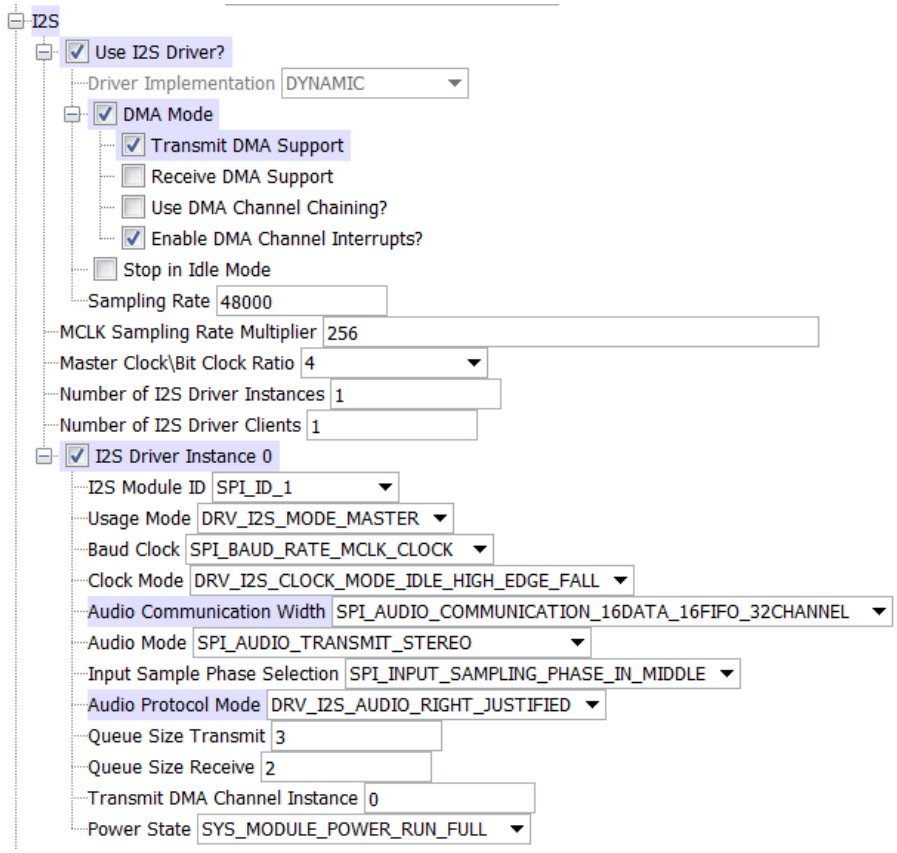
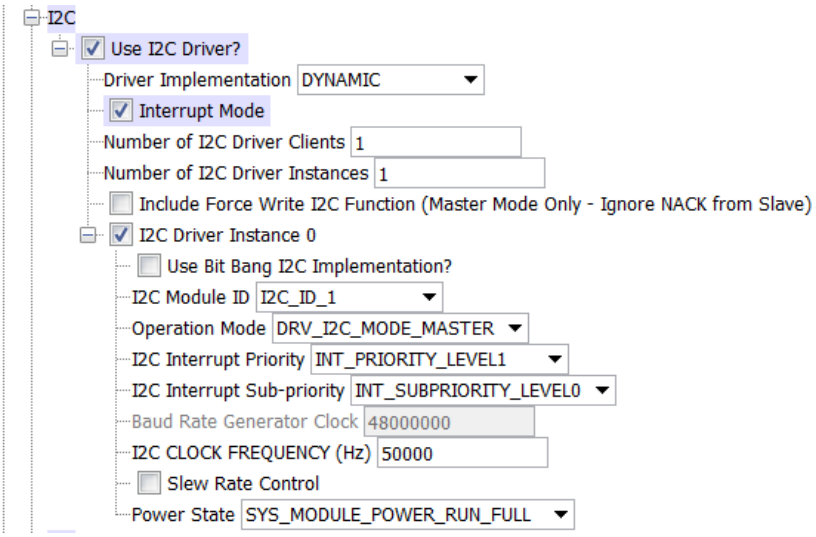


Figure 3: I2C Driver MHC Configuration



Migrating the AK4642 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK4642 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK4642 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4642 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4642`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak4642.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak4642.c`	This file contains implementation of the AK4642 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
There are no optional files for this driver.	N/A

Module Dependencies

The AK4642 Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4642_Initialize	Initializes hardware and data for the instance of the AK4642 DAC module
	DRV_AK4642_Deinitialize	Deinitializes the specified instance of the AK4642 driver module
	DRV_AK4642_Status	Gets the current status of the AK4642 driver module.
	DRV_AK4642_Tasks	Maintains the driver's control and data interface state machine.

b) Client Setup Functions

	Name	Description
	DRV_AK4642_Open	Opens the specified AK4642 driver instance and returns a handle to it
	DRV_AK4642_Close	Closes an opened-instance of the AK4642 driver

c) Codec Specific Functions

	Name	Description
	DRV_AK4642_MuteOff	This function disables AK4642 output for soft mute.
	DRV_AK4642_MuteOn	This function allows AK4642 output for soft mute on.
	DRV_AK4642_SamplingRateGet	This function gets the sampling rate set on the AK4642. **Implementation:** Dynamic
	DRV_AK4642_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK4642_VolumeGet	This function gets the volume for AK4642 CODEC.
	DRV_AK4642_VolumeSet	This function sets the volume for AK4642 CODEC.
	DRV_AK4642_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4642_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4642_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK4642_MicSet	This function select the single-ended AK4642 microphone input for the AK4642 Codec

d) Data Transfer Functions

	Name	Description
	DRV_AK4642_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK4642_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4642_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4642_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

e) Other Functions

	Name	Description
	DRV_AK4642_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK4642_VersionGet	This function returns the version of AK4642 driver
	DRV_AK4642_VersionStrGet	This function returns the version of AK4642 driver in string format.

f) Data Types and Constants

	Name	Description
	_DRV_AK4642_H	Include files.
	DRV_AK4642_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4642_COUNT	Number of valid AK4642 driver indices
	DRV_AK4642_INDEX_0	AK4642 driver index definitions
	DRV_AK4642_INDEX_1	This is macro DRV_AK4642_INDEX_1.
	DRV_AK4642_INDEX_2	This is macro DRV_AK4642_INDEX_2.
	DRV_AK4642_INDEX_3	This is macro DRV_AK4642_INDEX_3.
	DRV_AK4642_INDEX_4	This is macro DRV_AK4642_INDEX_4.
	DRV_AK4642_INDEX_5	This is macro DRV_AK4642_INDEX_5.
	DRV_AK4642_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4642_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4642_BUFFER_EVENT_HANDLER	Pointer to a AK4642 Driver Buffer Event handler function
	DRV_AK4642_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4642_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4642_COMMAND_EVENT_HANDLER	Pointer to a AK4642 Driver Command Event Handler Function
	DRV_AK4642_INIT	Defines the data required to initialize or reinitialize the AK4642 driver
	DRV_AK4642_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4642_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.
	DRV_AK4642_MIC	This is type DRV_AK4642_MIC.

Description

This section describes the API functions of the AK4642 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4642_Initialize Function

Initializes hardware and data for the instance of the AK4642 DAC module

File

drv_ak4642.h

C

```
SYS_MODULE_OBJ DRV_AK4642_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4642 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4642 routine is called.

This routine should only be called once during system initialization unless DRV_AK4642_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. DRV_I2C_Initialize must be called if SPI driver is used for handling the control interface of this CODEC driver.

Example

```
DRV_AK4642_INIT                 init;
SYS_MODULE_OBJ                  objectHandle;

init->inUse                             = true;
init->status                            = SYS_STATUS_BUSY;
init->numClients                        = 0;
init->i2sDriverModuleIndex              = ak4642Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex              = ak4642Init->i2cDriverModuleIndex;
init->samplingRate                      = DRV_AK4642_AUDIO_SAMPLING_RATE;
init->audioDataFormat                   = DRV_AK4642_AUDIO_DATA_FORMAT_MACRO;

init->isInInterruptContext              = false;

init->commandCompleteCallback = (DRV_AK4642_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;
init->mclk_multiplier = DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK4642_Initialize(DRV_AK4642_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK4642_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT *const init

);

DRV_AK4642_Deinitialize Function

Deinitializes the specified instance of the AK4642 driver module

File

drv_ak4642.h

C

```
void DRV_AK4642_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4642 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4642_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4642_Initialize
SYS_STATUS          status;


DRV_AK4642_Deinitialize(object);

status = DRV_AK4642_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

void DRV_AK4642_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4642_Status Function

Gets the current status of the AK4642 driver module.

File

drv_ak4642.h

C

```
SYS_STATUS DRV_AK4642_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4642 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4642_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4642_Initialize
SYS_STATUS          AK4642Status;
```

```
AK4642Status = DRV_AK4642_Status(object);
if (SYS_STATUS_READY == AK4642Status)
{
    // This means the driver can be opened using the
    // DRV_AK4642_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

SYS_STATUS DRV_AK4642_Status(SYS_MODULE_OBJ object)

DRV_AK4642_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_ak4642.h

C

```
void DRV_AK4642_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4642_Initialize

while (true)
{
    DRV_AK4642_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4642_Initialize)

Function

void DRV_AK4642_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_AK4642_Open Function

Opens the specified AK4642 driver instance and returns a handle to it

File

drv_ak4642.h

C

```
DRV_HANDLE DRV_AK4642_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK4642_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified AK4642 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

AK4642 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4642_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4642_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4642_Open(DRV_AK4642_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4642_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4642_Close Function

Closes an opened-instance of the AK4642 driver

File

drv_ak4642.h

C

```
void DRV_AK4642_Close(const DRV_HANDLE handle);
```

Returns

- None

Description

This routine closes an opened-instance of the AK4642 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4642_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4642_Open

DRV_AK4642_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_Close(DRV_Handle handle)

c) Codec Specific Functions

DRV_AK4642_MuteOff Function

This function disables AK4642 output for soft mute.

File

drv_ak4642.h

C

```
void DRV_AK4642_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4642 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

    DRV_AK4642_MuteOff(myAK4642Handle); //AK4642 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_MuteOff(DRV_HANDLE handle)

DRV_AK4642_MuteOn Function

This function allows AK4642 output for soft mute on.

File

drv_ak4642.h

C

```
void DRV_AK4642_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4642 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

DRV_AK4642_MuteOn(myAK4642Handle);  //AK4642 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_MuteOn(DRV_HANDLE handle);

DRV_AK4642_SamplingRateGet Function

This function gets the sampling rate set on the AK4642.

Implementation: Dynamic

File

drv_ak4642.h

C

```
uint32_t DRV_AK4642_SamplingRateGet(DRV_HANDLE handle);
```

Description

This function gets the sampling rate set on the DAC AK4642.

Remarks

None.

Example

```
uint32_t baudRate;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

baudRate = DRV_AK4642_SamplingRateGet(myAK4642Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4642_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4642_SamplingRateSet Function

This function sets the sampling rate of the media stream.

File

drv_ak4642.h

C

```
void DRV_AK4642_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

DRV_AK4642_SamplingRateSet(myAK4642Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
samplingRate	Sampling frequency in Hz

Function

void DRV_AK4642_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4642_VolumeGet Function

This function gets the volume for AK4642 CODEC.

File

drv_ak4642.h

C

```
uint8_t DRV_AK4642_VolumeGet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel);
```

Returns

None.

Description

This functions gets the current volume programmed to the CODEC AK4642.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

  volume = DRV_AK4642_VolumeGet(myAK4642Handle, DRV_AK4642_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified

Function

uint8_t DRV_AK4642_VolumeGet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel)

DRV_AK4642_VolumeSet Function

This function sets the volume for AK4642 CODEC.

File

drv_ak4642.h

C

```
void DRV_AK4642_VolumeSet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel, uint8_t volume);
```

Returns

None

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

    DRV_AK4642_VolumeSet(myAK4642Handle,DRV_AK4642_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_AK4642_VolumeSet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel, uint8_t volume);

DRV_AK4642_IntExtMicSet Function

This function sets up the codec for the internal or the external microphone use.

File

drv_ak4642.h

C

```
void DRV_AK4642_IntExtMicSet(DRV_HANDLE handle, DRV_AK4642_INT_EXT_MIC micInput);
```

Returns

None

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	INT_MIC or EXT_MIC

Function

void DRV_AK4642_IntExtMicSet(DRV_HANDLE handle,
DRV_AK4642_INT_EXT_MIC micInput);

DRV_AK4642_MonoStereoMicSet Function

This function sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak4642.h

C

```
void DRV_AK4642_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK4642_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

Currently the ak4642 codec does not work in the MONO_LEFT_CHANNEL mode. This issue will be followed up with AKM.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.
DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
mono_stereo_mic	Mono / Stereo mic setup

Function

void DRV_AK4642_MonoStereoMicSet(DRV_HANDLE handle);

DRV_AK4642_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_ak4642.h

C

```
void DRV_AK4642_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.
DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_AK4642_SetAudioCommunicationMode
(
DRV_HANDLE handle,
const DATA_LENGTH dl,
const SAMPLE_LENGTH sl
)

DRV_AK4642_MicSet Function

This function select the single-ended AK4642 microphone input for the AK4642 Codec

File

drv_ak4642.h

C

```
void DRV_AK4642_MicSet(DRV_HANDLE handle, DRV_AK4642_MIC micInput);
```

Returns

None

Description

This function selects the single-ended AK4642 microphone input for the AK4642 Codec (Where the MEMS mic is MIC1, and the external Microphone input is MIC2 on the AK4642 XC32 Daughter Board)

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	MIC1 or MIC2

Function

void DRV_AK4642_MicSet(DRV_HANDLE handle, DRV_AK4642_MIC micInput);

d) Data Transfer Functions

DRV_AK4642_BufferAddWrite Function

Schedule a non-blocking driver write operation.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4642_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myAK4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWrite(myAK4642handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:
```

```
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as return by the DRV_AK4642_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4642_BufferAddWrite

(

const DRV_HANDLE handle,

DRV_AK4642_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK4642_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddRead(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK4642_Open call.

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as return by the DRV_AK4642_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4642_BufferAddRead
(
const DRV_HANDLE handle,
DRV_AK4642_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4642_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void * transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every AK4642 write. The transmit and receive size must be same.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_AK4642_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
```

```
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myak4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myak4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWriteRead(myak4642handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as returned by the DRV_AK4642_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_AK4642_BufferAddWriteRead
(
const DRV_HANDLE handle,
DRV_AK4642_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)

DRV_AK4642_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4642_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4642_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myAK4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWrite(myAK4642handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;
```

```
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4642_BufferEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4642_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

e) Other Functions

DRV_AK4642_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

File

drv_ak4642.h

C

```
void DRV_AK4642_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4642_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4642_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4642 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
```

```
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_CommandEventHandlerSet(myAK4642Handle,
                APP_AK4642CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_DeEmphasisFilterSet(myAK4642Handle, DRV_AK4642_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4642CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4642_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4642_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4642_VersionGet Function

This function returns the version of AK4642 driver

File

drv_ak4642.h

C

```
uint32_t DRV_AK4642_VersionGet();
```

Returns

returns the version of AK4642 driver.

Description

The version number returned from the DRV_AK4642_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t AK4642version;
AK4642version = DRV_AK4642_VersionGet();
```

Function

uint32_t DRV_AK4642_VersionGet(void)

DRV_AK4642_VersionStrGet Function

This function returns the version of AK4642 driver in string format.

File

drv_ak4642.h

C

```
int8_t* DRV_AK4642_VersionStrGet();
```

Returns

returns a string containing the version of AK4642 driver.

Description

The DRV_AK4642_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4642 driver's version number. is the AK4642 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces. For example, "0.03a" "1.00"

Remarks

None

Preconditions

None.

Example

```
int8_t *AK4642string;
AK4642string = DRV_AK4642_VersionStrGet();
```

Function

int8_t* DRV_AK4642_VersionStrGet(void)

f) Data Types and Constants

_DRV_AK4642_H Macro

File

drv_ak4642.h

C

```
#define _DRV_AK4642_H
```

Description

Include files.

DRV_AK4642_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4642.h

C

```
#define DRV_AK4642_BUFFER_HANDLE_INVALID ((DRV_AK4642_BUFFER_HANDLE)(-1))
```

Description

AK4642 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4642_BufferAddWrite() and the DRV_AK4642_BufferAddRead() function if the buffer add request was not successful.

Remarks

None.

DRV_AK4642_COUNT Macro

Number of valid AK4642 driver indices

File

drv_ak4642.h

C

```
#define DRV_AK4642_COUNT
```

Description

AK4642 Driver Module Count

This constant identifies the maximum number of AK4642 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4642 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_AK4642_INDEX_0 Macro

AK4642 driver index definitions

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_0 0
```

Description

Driver AK4642 Module Index

These constants provide AK4642 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4642_Initialize and DRV_AK4642_Open routines to identify the driver instance in use.

DRV_AK4642_INDEX_1 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_1 1
```

Description

This is macro DRV_AK4642_INDEX_1.

DRV_AK4642_INDEX_2 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_2 2
```

Description

This is macro DRV_AK4642_INDEX_2.

DRV_AK4642_INDEX_3 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_3 3
```

Description

This is macro DRV_AK4642_INDEX_3.

DRV_AK4642_INDEX_4 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_4 4
```

Description

This is macro DRV_AK4642_INDEX_4.

DRV_AK4642_INDEX_5 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_5 5
```

Description

This is macro DRV_AK4642_INDEX_5.

DRV_AK4642_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_AUDIO_DATA_FORMAT_NOT_APPLICABLE = 0,
  DRV_AK4642_AUDIO_DATA_FORMAT_16BITMSB_SDTO_16BITLSB_SDTI,
  DRV_AK4642_AUDIO_DATA_FORMAT_16BITMSB_SDTO_16BITMSB_SDTI,
  DRV_AK4642_AUDIO_DATA_FORMAT_I2S
```

```
} DRV_AK4642_AUDIO_DATA_FORMAT;
```

Description

AK4642 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_AK4642_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_BUFFER_EVENT_COMPLETE,
  DRV_AK4642_BUFFER_EVENT_ERROR,
  DRV_AK4642_BUFFER_EVENT_ABORT
} DRV_AK4642_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4642_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4642_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4642_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4642 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4642_BufferAddWrite() or the DRV_AK4642_BufferAddRead() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4642_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4642_BUFFER_EVENT_HANDLER Type

Pointer to a AK4642 Driver Buffer Event handler function

File

drv_ak4642.h

C

```
typedef void (* DRV_AK4642_BUFFER_EVENT_HANDLER)(DRV_AK4642_BUFFER_EVENT event, DRV_AK4642_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4642 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4642 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4642_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4642_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4642_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4642_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver(i2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4642_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4642_BUFFER_EVENT event,
                               DRV_AK4642_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4642_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4642.h

C

```
typedef uintptr_t DRV_AK4642_BUFFER_HANDLE;
```

Description

AK4642 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4642_BufferAddWrite() or DRV_AK4642_BufferAddRead() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_AK4642_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_CHANNEL_LEFT,
  DRV_AK4642_CHANNEL_RIGHT,
  DRV_AK4642_CHANNEL_LEFT_RIGHT,
  DRV_AK4642_NUMBER_OF_CHANNELS
} DRV_AK4642_CHANNEL;
```

Description

AK4642 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4642_COMMAND_EVENT_HANDLER Type

Pointer to a AK4642 Driver Command Event Handler Function

File

drv_ak4642.h

C

```
typedef void (* DRV_AK4642_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4642 Driver Command Event Handler Function

This data type defines the required function signature for the AK4642 driver command event handling callback function.

A command is a control instruction to the AK4642 CODEC. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4642_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4642CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4642_INIT Structure

Defines the data required to initialize or reinitialize the AK4642 driver

File

drv_ak4642.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_AK4642_AUDIO_DATA_FORMAT audioDataFormat;
} DRV_AK4642_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_AK4642_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format

Description

AK4642 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4642 CODEC driver.

Remarks

None.

DRV_AK4642_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak4642.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK4642_INT_EXT_MIC;
```

Description

AK4642 Mic Internal / External Input

This enumeration identifies the Mic input source.

DRV_AK4642_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono / Stereo.

File

drv_ak4642.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
```

```
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK4642_MONO_STEREO_MIC;
```

Description

AK4642 Mic Mono / Stereo Input

This enumeration identifies the Mic input as Mono / Stereo.

DRV_AK4642_MIC Enumeration

File

drv_ak4642.h

C

```
typedef enum {
  MIC1 = 0,
  MIC2,
  DRV_AK4642_NUMBER_MIC
} DRV_AK4642_MIC;
```

Members

Members	Description
MIC1 = 0	INT_MIC
MIC2	EXT_MIC

Description

This is type DRV_AK4642_MIC.

Files

Files

Name	Description
drv_ak4642.h	AK4642 CODEC Driver Interface header file
drv_ak4642_config_template.h	AK4642 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4642 Codec Driver Library.

drv_ak4642.h

AK4642 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4642_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4642_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4642_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4642_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4642_MIC	This is type DRV_AK4642_MIC.
	DRV_AK4642_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Functions

	Name	Description
	DRV_AK4642_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4642_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK4642_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4642_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

	DRV_AK4642_Close	Closes an opened-instance of the AK4642 driver
	DRV_AK4642_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK4642_Deinitialize	Deinitializes the specified instance of the AK4642 driver module
	DRV_AK4642_Initialize	Initializes hardware and data for the instance of the AK4642 DAC module
	DRV_AK4642_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4642_MicSet	This function select the single-ended AK4642 microphone input for the AK4642 Codec
	DRV_AK4642_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4642_MuteOff	This function disables AK4642 output for soft mute.
	DRV_AK4642_MuteOn	This function allows AK4642 output for soft mute on.
	DRV_AK4642_Open	Opens the specified AK4642 driver instance and returns a handle to it
	DRV_AK4642_SamplingRateGet	This function gets the sampling rate set on the AK4642. **Implementation:** Dynamic
	DRV_AK4642_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK4642_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK4642_Status	Gets the current status of the AK4642 driver module.
	DRV_AK4642_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK4642_VersionGet	This function returns the version of AK4642 driver
	DRV_AK4642_VersionStrGet	This function returns the version of AK4642 driver in string format.
	DRV_AK4642_VolumeGet	This function gets the volume for AK4642 CODEC.
	DRV_AK4642_VolumeSet	This function sets the volume for AK4642 CODEC.

Macros

	Name	Description
	_DRV_AK4642_H	Include files.
	DRV_AK4642_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4642_COUNT	Number of valid AK4642 driver indices
	DRV_AK4642_INDEX_0	AK4642 driver index definitions
	DRV_AK4642_INDEX_1	This is macro DRV_AK4642_INDEX_1.
	DRV_AK4642_INDEX_2	This is macro DRV_AK4642_INDEX_2.
	DRV_AK4642_INDEX_3	This is macro DRV_AK4642_INDEX_3.
	DRV_AK4642_INDEX_4	This is macro DRV_AK4642_INDEX_4.
	DRV_AK4642_INDEX_5	This is macro DRV_AK4642_INDEX_5.

Structures

	Name	Description
	DRV_AK4642_INIT	Defines the data required to initialize or reinitialize the AK4642 driver

Types

	Name	Description
	DRV_AK4642_BUFFER_EVENT_HANDLER	Pointer to a AK4642 Driver Buffer Event handler function
	DRV_AK4642_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4642_COMMAND_EVENT_HANDLER	Pointer to a AK4642 Driver Command Event Handler Function

Description

AK4642 CODEC Driver Interface

The AK4642 CODEC device driver interface provides a simple interface to manage the AK4642 16/24-Bit CODEC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4642 CODEC device driver.

File Name

drv_ak4642.h

Company

Microchip Technology Inc.

drv_ak4642_config_template.h

AK4642 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4642_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4642_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4642_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.

Description

AK4642 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4642_config_template.h

Company

Microchip Technology Inc.

AK4953 Codec Driver Library

This topic describes the AK4953 Codec Driver Library.

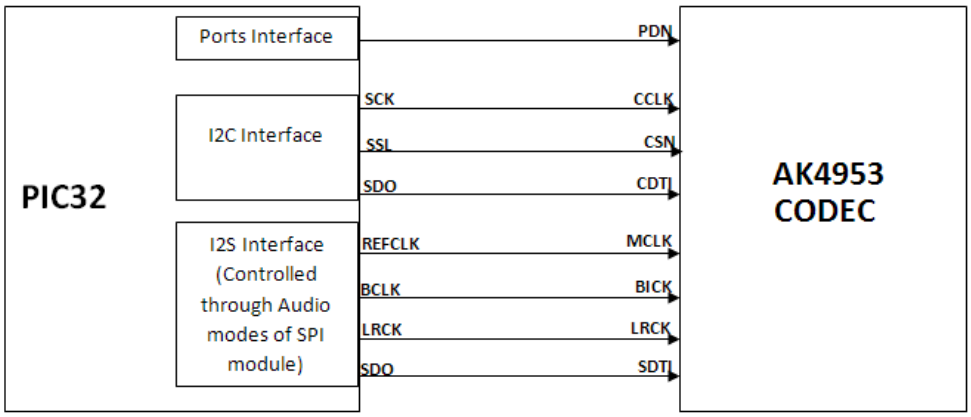
Introduction

This library provides an interface to manage the AK4953 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4953 module is 16/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK4953 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4953 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4953 Codec supports the following features:

- Audio Interface Format: MSB first
- ADC: 24-bit MSB justified, 16/24-bit I2S

- DAC: 24-bit MSB justified, 1-6bit LSB justified, 24-bit LSB justified, 16/24-bit I2S
- Sampling Frequency Range: 8 kHz to 192 kHz
- Digital Volume Control: +12dB ~ .115dB, 0.5dB Step
- SoftMute: On and Off
- Master Clock Frequencies: 32 fs/64 fs/128 fs/256 fs

Using the Library

This topic describes the basic architecture of the AK4953 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_AK4953.h`

The interface to the AK4953 Codec Driver library is defined in the `drv_AK4953.h` header file. Any C language source (`.c`) file that uses the AK4953 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4953 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4953 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4953 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Status Functions	Provides status functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4953 Codec Driver Library.

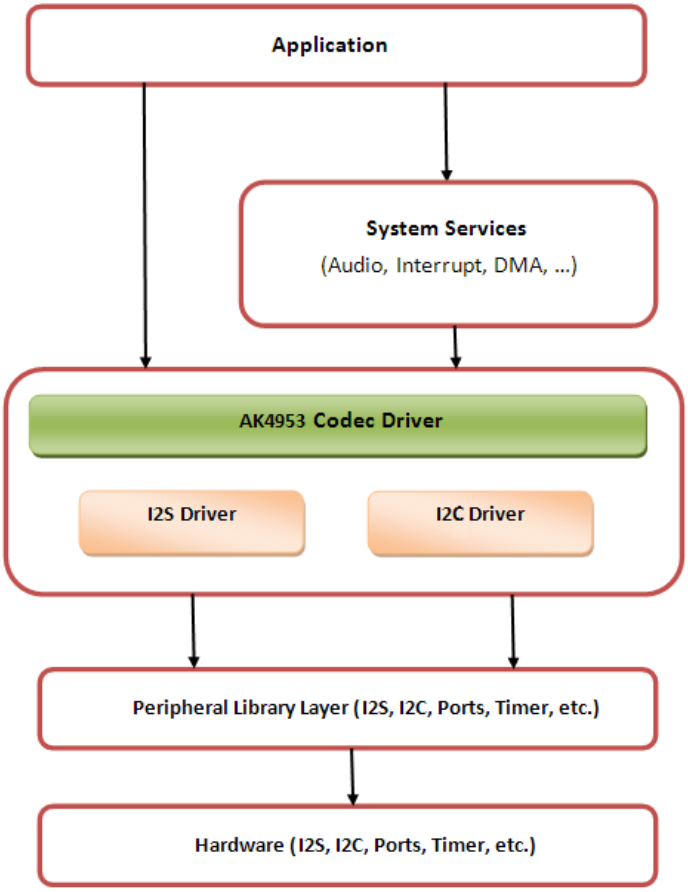
Abstraction Model

This library provides a low-level abstraction of the AK4953 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4953 Codec Driver is positioned in the MPLAB Harmony framework. The AK4953 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4953 module.

AK4953 Driver Abstraction Model



How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4953 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4953_INIT or by using Initialization Overrides) that are supported by the specific AK4953 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK4953_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4953_Deinitialize, DRV_ AK4953_Status and DRV_I2S_Tasks.

Implementations

The AK4953 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
DRV_AK4953_INIT drvak4953Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4953_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4953_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4953_VOLUME,
    .queueSizeTransmit = DRV_AK4953_TRANSMIT_QUEUE_SIZE,
};

// Initialize the I2C driver
   DRV_I2C0_Initialize();

// Initialize the I2S driver. The I2S module index should be same as the one used in initializing
// the I2S driver.
   sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)&drvI2S0InitData);

// Initialize the Codec driver
   sysObj.drvak4953Codec0 = DRV_AK4953_Initialize(DRV_AK4953_INDEX_0, (SYS_MODULE_INIT
*)&drvak4953Codec0InitData);

if (SYS_MODULE_OBJ_INVALID == AK4953DevObject)
{
// Handle error
}
```

Task Routine

The DRV_AK4953_Tasks will be called from the System Task Service.

Client Access

For the application to start using an instance of the module, it must call the DRV_AK4953_Open function. The DRV_AK4953_Open provides a driver handle to the AK4953 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4953_Deinitialize, the application must call the DRV_AK4953_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4953 Codec.

The following AK4953 Codec specific control command functions are provided:

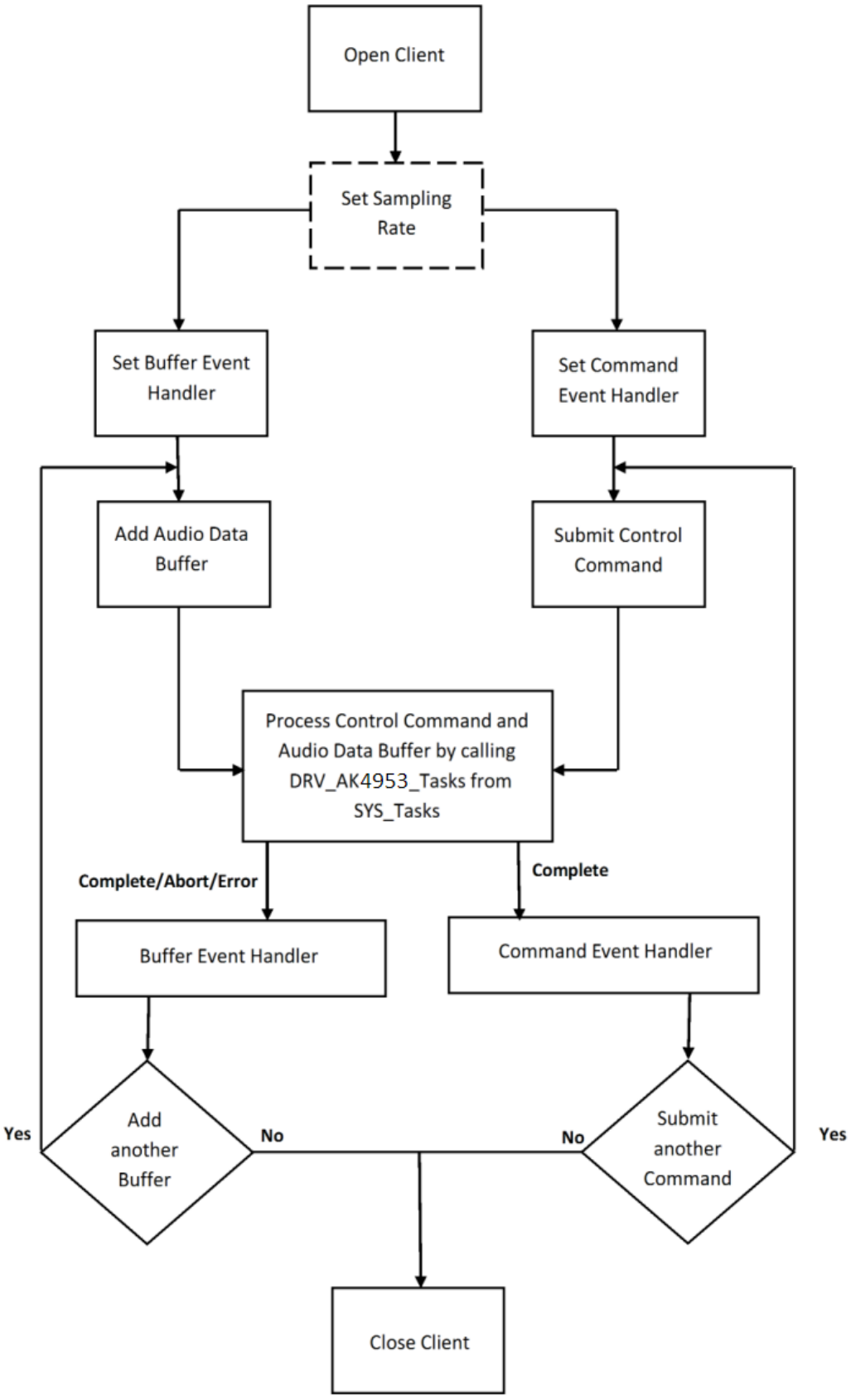
- DRV_AK4953_SamplingRateSet
- DRV_AK4953_SamplingRateGet
- DRV_AK4953_VolumeSet
- DRV_AK4953_VolumeGet
- DRV_AK4953_MuteOn
- DRV_AK4953_MuteOff
- DRV_AK4953_IntExtMicSet
- DRV_AK4953_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK4953 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_AK4953_BufferAddWrite, DRV_AK4953_BufferAddRead, and DRV_AK4953_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK4953_BUFFER_EVENT_COMPLETE, DRV_AK4953_BUFFER_EVENT_ERROR, or DRV_AK4953_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Macros

	Name	Description
	DRV_AK4953_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4953_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4953_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4953_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

The configuration of the AK4953 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4953 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4953 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4953_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_BCLK_BIT_CLK_DIVISOR
```

Description

AK4953 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

Following BCLK to LRCK ratios are supported 16bit data 16 bit channel :- 32fs, hence divisor would be 8 16bit data 32 bit channel :- 64fs, hence divisor would be 4

Remarks

None.

DRV_AK4953_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_CLIENTS_NUMBER DRV_AK4953_INSTANCES_NUMBER
```

Description

AK4953 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4953 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4953_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_INPUT_REFCLOCK
```

Description

AK4953 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4953_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_INSTANCES_NUMBER
```

Description

AK4953 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4953 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4953 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency I2S sampling frequency

Supported MCLK to Sampling frequency Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs

Remarks

None

DRV_AK4953_MCLK_SOURCE Macro

Indicate the input clock frequency to generate the MCLK to codec.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_MCLK_SOURCE
```

Description

AK4953 Data Interface Master Clock Speed configuration

Indicate the input clock frequency to generate the MCLK to codec.

Remarks

None.

DRV_AK4953_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_QUEUE_DEPTH_COMBINED
```

Description

AK4953 Driver Buffer Queue Entries

This macro defined the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_AK4953_BufferAddWrite function.

A buffer queue will contains buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all AK4953 driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

As an example, consider the case of a dynamic driver (say two instances) where instance one will queue up to three write requests and up to two read requests, and instance two will queue up to two write requests and up to six read requests, the value of this macro should be 13 (2 + 3 + 2 + 6).

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4953 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK4953 Codec Driver MHC Configuration

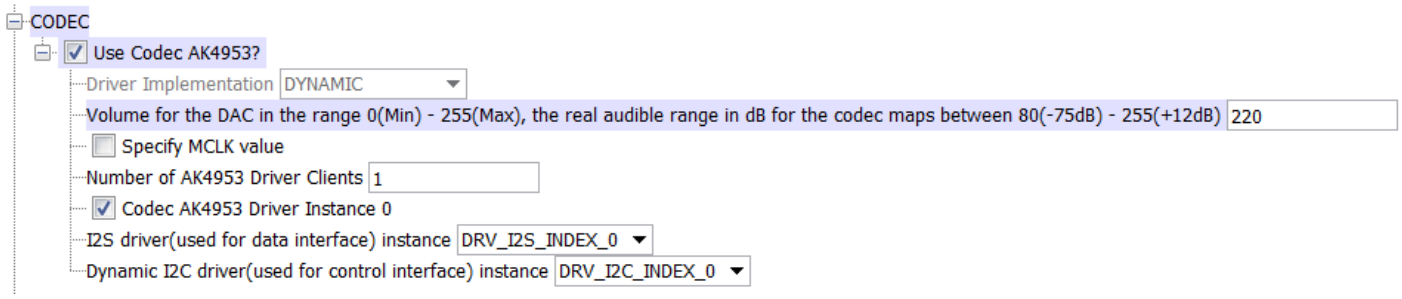


Figure 2: I2S Driver MHC Configuration

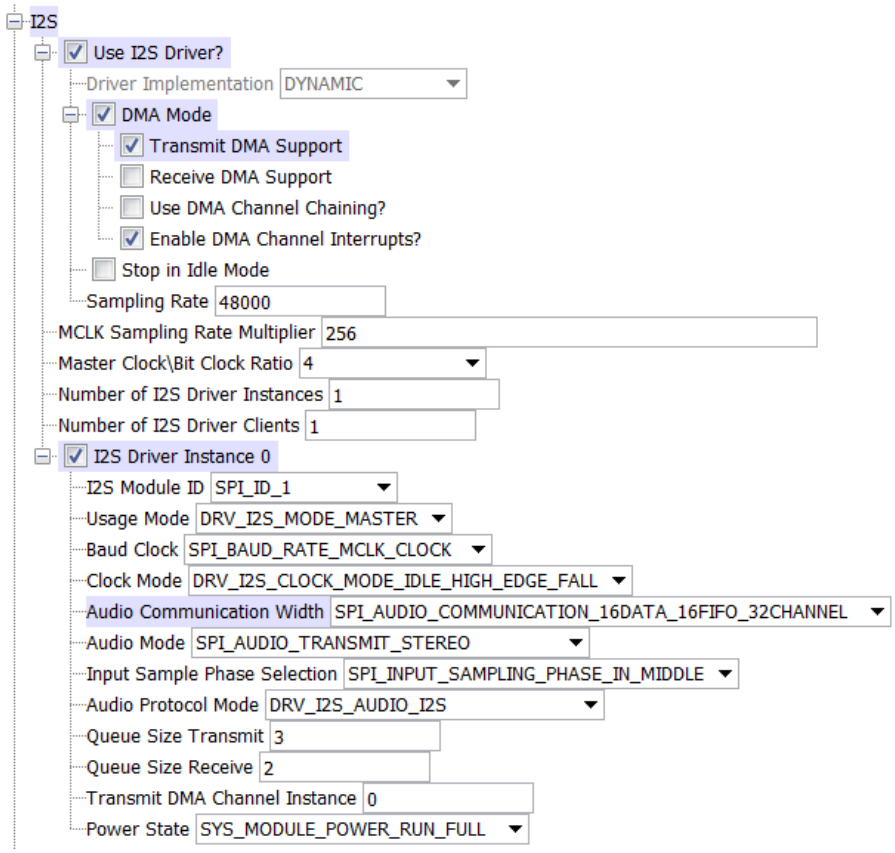
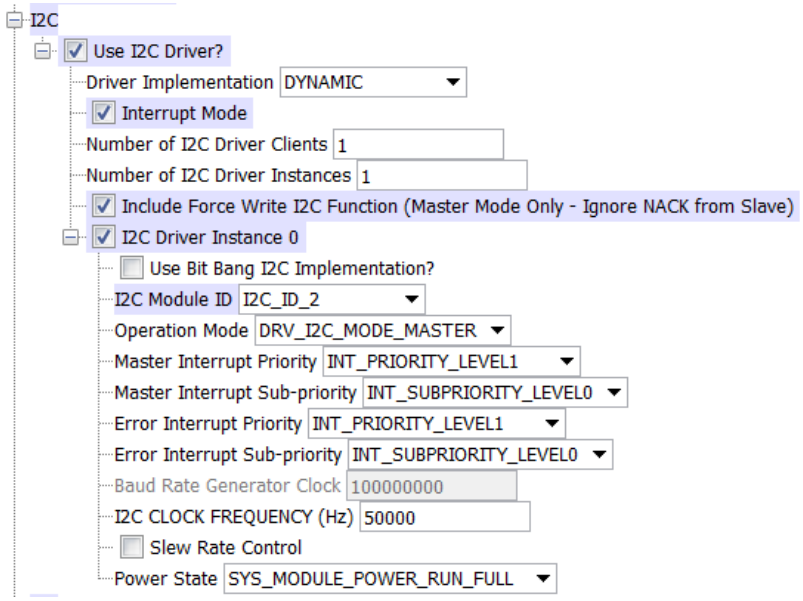


Figure 3: I2C Driver MHC Configuration



Migrating the AK4953 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK4953 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK4953 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4953 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4953`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_ak4953.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_ak4953.c	This file contains implementation of the AK4953 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AK4953 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4953_Initialize	Initializes hardware and data for the instance of the AK4953 DAC module. **Implementation:** Dynamic
	DRV_AK4953_Deinitialize	Deinitializes the specified instance of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Open	Opens the specified AK4953 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4953_Close	Closes an opened-instance of the AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4953_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4953_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4953_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4953_SetAudioCommunicationMode	This function provides a run time audio format configuration

b) Status Functions

	Name	Description
	DRV_AK4953_SamplingRateGet	This function gets the sampling rate set on the DAC AK4953. **Implementation:** Dynamic
	DRV_AK4953_Status	Gets the current status of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_VersionGet	This function returns the version of AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_VersionStrGet	This function returns the version of AK4953 driver in string format. **Implementation:** Dynamic
	DRV_AK4953_VolumeGet	This function gets the volume for AK4953 CODEC. **Implementation:** Dynamic

c) Other Functions

	Name	Description
	DRV_AK4953_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4953_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4953_MuteOff	This function disables AK4953 output for soft mute. **Implementation:** Dynamic
	DRV_AK4953_MuteOn	This function allows AK4953 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4953_VolumeSet	This function sets the volume for AK4953 CODEC. **Implementation:** Dynamic
	DRV_AK4953_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4953_IntExtMicSet	This function sets up the codec for the X32 DB internal or the external microphone use.
	DRV_AK4953_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4953_MicSet	This function sets up the codec for the internal or the AK4953 Mic1 or Mic2 input.

d) Data Types and Constants

	Name	Description
	DRV_AK4953_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4953_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4953_BUFFER_EVENT_HANDLER	Pointer to a AK4953 Driver Buffer Event handler function
	DRV_AK4953_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4953_COMMAND_EVENT_HANDLER	Pointer to a AK4953 Driver Command Event Handler Function
	DRV_AK4953_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4953_INIT	Defines the data required to initialize or reinitialize the AK4953 driver
	_DRV_AK4953_H	Include files.
	DRV_AK4953_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4953_COUNT	Number of valid AK4953 driver indices
	DRV_AK4953_INDEX_0	AK4953 driver index definitions
	DRV_AK4953_INDEX_1	This is macro DRV_AK4953_INDEX_1.
	DRV_AK4953_INDEX_2	This is macro DRV_AK4953_INDEX_2.
	DRV_AK4953_INDEX_3	This is macro DRV_AK4953_INDEX_3.
	DRV_AK4953_INDEX_4	This is macro DRV_AK4953_INDEX_4.
	DRV_AK4953_INDEX_5	This is macro DRV_AK4953_INDEX_5.
	DRV_AK4953_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4953_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4953_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.
	DRV_AK4953_MIC	This is type DRV_AK4953_MIC.

Description

This section describes the API functions of the AK4953 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4953_Initialize Function

Initializes hardware and data for the instance of the AK4953 DAC module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
SYS_MODULE_OBJ DRV_AK4953_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4953 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4953 routine is called.

This routine should only be called once during system initialization unless DRV_AK4953_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. Also DRV_I2C_Initialize must be called before calling this function to initialize the control interface of this CODEC driver.

Example

```
DRV_AK4953_INIT                 init;
SYS_MODULE_OBJ                  objectHandle;

init->inUse                             = true;
init->status                            = SYS_STATUS_BUSY;
init->numClients                        = 0;
init->i2sDriverModuleIndex              = ak4953Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex              = ak4953Init->i2cDriverModuleIndex;
init->samplingRate                      = DRV_AK4953_AUDIO_SAMPLING_RATE;
init->audioDataFormat                   = DRV_AK4953_AUDIO_DATA_FORMAT_MACRO;
for(index=0; index < DRV_AK4953_NUMBER_OF_CHANNELS; index++)
{
    init->volume[index] = ak4953Init->volume;
}
init->isInInterruptContext              = false;

init->commandCompleteCallback = (DRV_AK4953_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;

init->mclk_multiplier = DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK4953_Initialize(DRV_AK4953_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized

init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK4953_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT *const init
);

DRV_AK4953_Deinitialize Function

Deinitializes the specified instance of the AK4953 driver module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4953 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4953_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4953_Initialize
SYS_STATUS          status;


DRV_AK4953_Deinitialize(object);

status = DRV_AK4953_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4953_Initialize routine

Function

void DRV_AK4953_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4953_Open Function

Opens the specified AK4953 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ak4953.h

C

```
DRV_HANDLE DRV_AK4953_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK4953_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified AK4953 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

AK4953 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4953_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4953_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4953_Open(DRV_AK4953_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4953_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4953_Close Function

Closes an opened-instance of the AK4953 driver.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the AK4953 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4953_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4953_Open

DRV_AK4953_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_Close(DRV_Handle handle)

DRV_AK4953_Tasks Function

Maintains the driver's control and data interface state machine.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4953_Initialize

while (true)
{
    DRV_AK4953_Tasks (object);
```

```
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4953_Initialize)

Function

void DRV_AK4953_Tasks(SYS_MODULE_OBJ object);

DRV_AK4953_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4953_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4953_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4953 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_CommandEventHandlerSet(myAK4953Handle,
                APP_AK4953CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_DeEmphasisFilterSet(myAK4953Handle, DRV_AK4953_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4953CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4953_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4953_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4953_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4953_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4953_BufferAddRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myAK4953Handle,
```

```
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddRead(myAK4953handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4953_BufferEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4953_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4953_SamplingRateSet Function

This function sets the sampling rate of the media stream.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

DRV_AK4953_SamplingRateSet(myAK4953Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4953_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_ak4953.h

C

```
void DRV_AK4953_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_AK4953_SetAudioCommunicationMode

(

DRV_HANDLE handle,

const DATA_LENGTH dl,

const SAMPLE_LENGTH sl
)

b) Status Functions

DRV_AK4953_SamplingRateGet Function

This function gets the sampling rate set on the DAC AK4953.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint32_t DRV_AK4953_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

This function gets the sampling rate set on the DAC AK4953.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t baudRate;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

baudRate = DRV_AK4953_SamplingRateGet(myAK4953Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4953_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4953_Status Function

Gets the current status of the AK4953 driver module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
SYS_STATUS DRV_AK4953_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4953 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4953_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4953_Initialize
SYS_STATUS          AK4953Status;

AK4953Status = DRV_AK4953_Status(object);
if (SYS_STATUS_READY == AK4953Status)
{
    // This means the driver can be opened using the
    // DRV_AK4953_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4953_Initialize routine

Function

SYS_STATUS DRV_AK4953_Status(SYS_MODULE_OBJ object)

DRV_AK4953_VersionGet Function

This function returns the version of AK4953 driver.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint32_t DRV_AK4953_VersionGet();
```

Returns

returns the version of AK4953 driver.

Description

The version number returned from the DRV_AK4953_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
    uint32_t AK4953version;
    AK4953version = DRV_AK4953_VersionGet();
```

Function

uint32_t DRV_AK4953_VersionGet(void)

DRV_AK4953_VersionStrGet Function

This function returns the version of AK4953 driver in string format.

Implementation: Dynamic

File

drv_ak4953.h

C

```
int8_t* DRV_AK4953_VersionStrGet();
```

Returns

returns a string containing the version of AK4953 driver.

Description

The DRV_AK4953_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4953 driver's version number. is the AK4953 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces.

Remarks

None.

Preconditions

None.

Example 1

"0.03a" "1.00"

Example 2

```
int8_t *AK4953string;
AK4953string = DRV_AK4953_VersionStrGet();
```

Function

int8_t* DRV_AK4953_VersionStrGet(void)

DRV_AK4953_VolumeGet Function

This function gets the volume for AK4953 CODEC.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint8_t DRV_AK4953_VolumeGet(DRV_HANDLE handle, DRV_AK4953_CHANNEL chan);
```

Returns

None.

Description

This functions gets the current volume programmed to the CODEC AK4953.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

  volume = DRV_AK4953_VolumeGet(myAK4953Handle,DRV_AK4953_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set

Function

uint8_t DRV_AK4953_VolumeGet(DRV_HANDLE handle, DRV_AK4953_CHANNEL chan)

c) Other Functions

DRV_AK4953_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4953_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myAK4953Handle,
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddWrite(myAK4953handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as return by the DRV_AK4953_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4953_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK4953_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4953_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every AK4953 write. The transmit and receive size must be same.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_AK4953_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myak4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myak4953Handle,
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddWriteRead(myak4953handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
```

```
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as returned by the DRV_AK4953_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_AK4953_BufferAddWriteRead
(
const DRV_HANDLE handle,
DRV_AK4953_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)

DRV_AK4953_MuteOff Function

This function disables AK4953 output for soft mute.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4953 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

    DRV_AK4953_MuteOff(myAK4953Handle); //AK4953 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MuteOff(DRV_HANDLE handle)

DRV_AK4953_MuteOn Function

This function allows AK4953 output for soft mute on.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4953 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

DRV_AK4953_MuteOn(myAK4953Handle);  //AK4953 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MuteOn(DRV_HANDLE handle);

DRV_AK4953_VolumeSet Function

This function sets the volume for AK4953 CODEC.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_VolumeSet(DRV_HANDLE handle, DRV_AK4953_CHANNEL channel, uint8_t volume);
```

Returns

None.

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

    DRV_AK4953_VolumeSet(myAK4953Handle, DRV_AK4953_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_AK4953_VolumeSet(DRV_HANDLE handle, DRV_AK4953_CHANNEL channel, uint8_t volume);

DRV_AK4953_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddRead(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK4953_Open call.

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as return by the DRV_AK4953_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

```
void DRV_AK4953_BufferAddRead
(
const       DRV_HANDLE handle,
      DRV_AK4953_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)
```

DRV_AK4953_IntExtMicSet Function

This function sets up the codec for the X32 DB internal or the external microphone use.

File

drv_ak4953.h

C

```
void DRV_AK4953_IntExtMicSet(DRV_HANDLE handle, DRV_AK4953_INT_EXT_MIC micInput);
```

Returns

None

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4953_IntExtMicSet

DRV_AK4953_MonoStereoMicSet Function

This function sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak4953.h

C

```
void DRV_AK4953_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK4953_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.
DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MonoStereoMicSet(DRV_HANDLE handle);

DRV_AK4953_MicSet Function

This function sets up the codec for the internal or the AK4953 Mic1 or Mic2 input.

File

drv_ak4953.h

C

```
void DRV_AK4953_MicSet(DRV_HANDLE handle, DRV_AK4953_MIC micInput);
```

Returns

None

Description

This function sets up the codec.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.
DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4953_IntMic12Set

d) Data Types and Constants

DRV_AK4953_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_LSB_SDTI = 0,
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_16BIT_LSB_SDTI,
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_MSB_SDTI,
  DRV_AK4953_AUDIO_DATA_FORMAT_I2S
} DRV_AK4953_AUDIO_DATA_FORMAT;
```

Description

AK4953 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_AK4953_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_BUFFER_EVENT_COMPLETE,
  DRV_AK4953_BUFFER_EVENT_ERROR,
  DRV_AK4953_BUFFER_EVENT_ABORT
} DRV_AK4953_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4953_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4953_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4953_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4953 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4953_BufferAddWrite() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4953_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4953_BUFFER_EVENT_HANDLER Type

Pointer to a AK4953 Driver Buffer Event handler function

File

drv_ak4953.h

C

```
typedef void (* DRV_AK4953_BUFFER_EVENT_HANDLER)(DRV_AK4953_BUFFER_EVENT event, DRV_AK4953_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4953 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4953 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4953_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4953_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4953_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4953_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4953_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4953_BUFFER_EVENT event,
                               DRV_AK4953_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4953_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4953.h

C

```
typedef uintptr_t DRV_AK4953_BUFFER_HANDLE;
```

Description

AK4953 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4953_BufferAddWrite() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_AK4953_COMMAND_EVENT_HANDLER Type

Pointer to a AK4953 Driver Command Event Handler Function

File

drv_ak4953.h

C

```
typedef void (* DRV_AK4953_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4953 Driver Command Event Handler Function

This data type defines the required function signature for the AK4953 driver command event handling callback function.

A command is a control instruction to the AK4953 CODEC. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4953_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4953CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4953_DIGITAL_BLOCK_CONTROL Enumeration

Identifies Bass-Boost Control function

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_RECORDING_MODE,
  DRV_AK4953_PLAYBACK_MODE,
  DRV_AK4953_RECORDING_PLAYBACK_2_MODE,
  DRV_AK4953_LOOPBACK_MODE
} DRV_AK4953_DIGITAL_BLOCK_CONTROL;
```

Members

Members	Description
DRV_AK4953_RECORDING_MODE	This is the default setting
DRV_AK4953_PLAYBACK_MODE	Min control
DRV_AK4953_RECORDING_PLAYBACK_2_MODE	Medium control
DRV_AK4953_LOOPBACK_MODE	Maximum control

Description

AK4953 Bass-Boost Control

This enumeration identifies the settings for Bass-Boost Control function.

Remarks

None.

DRV_AK4953_INIT Structure

Defines the data required to initialize or reinitialize the AK4953 driver

File

drv_ak4953.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_AK4953_AUDIO_DATA_FORMAT audioDataFormat;
} DRV_AK4953_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_AK4953_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format

Description

AK4953 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4953 CODEC driver.

Remarks

None.

_DRV_AK4953_H Macro

File

drv_ak4953.h

C

```
#define _DRV_AK4953_H
```

Description

Include files.

DRV_AK4953_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4953.h

C

```
#define DRV_AK4953_BUFFER_HANDLE_INVALID ((DRV_AK4953_BUFFER_HANDLE)(-1))
```

Description

AK4953 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4953_BufferAddWrite() function if the buffer add request was not successful.

Remarks

None

DRV_AK4953_COUNT Macro

Number of valid AK4953 driver indices

File

drv_ak4953.h

C

```
#define DRV_AK4953_COUNT
```

Description

AK4953 Driver Module Count

This constant identifies the maximum number of AK4953 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4953 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_AK4953_INDEX_0 Macro

AK4953 driver index definitions

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_0 0
```

Description

Driver AK4953 Module Index

These constants provide AK4953 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4953_Initialize and DRV_AK4953_Open routines to identify the driver instance in use.

DRV_AK4953_INDEX_1 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_1 1
```

Description

This is macro DRV_AK4953_INDEX_1.

DRV_AK4953_INDEX_2 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_2 2
```

Description

This is macro DRV_AK4953_INDEX_2.

DRV_AK4953_INDEX_3 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_3 3
```

Description

This is macro DRV_AK4953_INDEX_3.

DRV_AK4953_INDEX_4 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_4 4
```

Description

This is macro DRV_AK4953_INDEX_4.

DRV_AK4953_INDEX_5 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_5 5
```

Description

This is macro DRV_AK4953_INDEX_5.

DRV_AK4953_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_CHANNEL_LEFT,
  DRV_AK4953_CHANNEL_RIGHT,
  DRV_AK4953_CHANNEL_LEFT_RIGHT,
  DRV_AK4953_NUMBER_OF_CHANNELS
} DRV_AK4953_CHANNEL;
```

Description

AK4953 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4953_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak4953.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK4953_INT_EXT_MIC;
```

Description

AK4953 Mic Internal / External Input

This enumeration identifies the Mic input source.

DRV_AK4953_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono / Stereo.

File

drv_ak4953.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
```

```
  STEREO
} DRV_AK4953_MONO_STEREO_MIC;
```

Description

AK4953 Mic Mono / Stereo Input

This enumeration identifies the Mic input as Mono / Stereo.

DRV_AK4953_MIC Enumeration

File

drv_ak4953.h

C

```
typedef enum {
  MIC1 = 0,
  MIC2,
  MIC3,
  DRV_AK4953_NUMBER_OF_MIC
} DRV_AK4953_MIC;
```

Members

Members	Description
MIC1 = 0	INT_MIC
MIC2	EXT_MIC
MIC3	LINE-IN

Description

This is type DRV_AK4953_MIC.

Files

Files

Name	Description
drv_ak4953.h	AK4953 CODEC Driver Interface header file
drv_ak4953_config_template.h	AK4953 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4953Codec Driver Library.

drv_ak4953.h

AK4953 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4953_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4953_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4953_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4953_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4953_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4953_MIC	This is type DRV_AK4953_MIC.
	DRV_AK4953_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Functions

	Name	Description
	DRV_AK4953_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4953_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic

	DRV_AK4953_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4953_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4953_Close	Closes an opened-instance of the AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4953_Deinitialize	Deinitializes the specified instance of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Initialize	Initializes hardware and data for the instance of the AK4953 DAC module. **Implementation:** Dynamic
	DRV_AK4953_IntExtMicSet	This function sets up the codec for the X32 DB internal or the external microphone use.
	DRV_AK4953_MicSet	This function sets up the codec for the internal or the AK4953 Mic1 or Mic2 input.
	DRV_AK4953_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4953_MuteOff	This function disables AK4953 output for soft mute. **Implementation:** Dynamic
	DRV_AK4953_MuteOn	This function allows AK4953 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4953_Open	Opens the specified AK4953 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4953_SamplingRateGet	This function gets the sampling rate set on the DAC AK4953. **Implementation:** Dynamic
	DRV_AK4953_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4953_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK4953_Status	Gets the current status of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4953_VersionGet	This function returns the version of AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_VersionStrGet	This function returns the version of AK4953 driver in string format. **Implementation:** Dynamic
	DRV_AK4953_VolumeGet	This function gets the volume for AK4953 CODEC. **Implementation:** Dynamic
	DRV_AK4953_VolumeSet	This function sets the volume for AK4953 CODEC. **Implementation:** Dynamic

Macros

	Name	Description
	_DRV_AK4953_H	Include files.
	DRV_AK4953_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4953_COUNT	Number of valid AK4953 driver indices
	DRV_AK4953_INDEX_0	AK4953 driver index definitions
	DRV_AK4953_INDEX_1	This is macro DRV_AK4953_INDEX_1.
	DRV_AK4953_INDEX_2	This is macro DRV_AK4953_INDEX_2.
	DRV_AK4953_INDEX_3	This is macro DRV_AK4953_INDEX_3.
	DRV_AK4953_INDEX_4	This is macro DRV_AK4953_INDEX_4.
	DRV_AK4953_INDEX_5	This is macro DRV_AK4953_INDEX_5.

Structures

	Name	Description
	DRV_AK4953_INIT	Defines the data required to initialize or reinitialize the AK4953 driver

Types

	Name	Description
	DRV_AK4953_BUFFER_EVENT_HANDLER	Pointer to a AK4953 Driver Buffer Event handler function
	DRV_AK4953_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4953_COMMAND_EVENT_HANDLER	Pointer to a AK4953 Driver Command Event Handler Function

Description

AK4953 CODEC Driver Interface

The AK4953 CODEC device driver interface provides a simple interface to manage the AK4953 106dB 192kHz 24-Bit DAC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4953 CODEC device driver.

File Name

drv_AK4953.h

Company

Microchip Technology Inc.

drv_ak4953_config_template.h

AK4953 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4953_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4953_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4953_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4953_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

AK4953 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4953_config_template.h

Company

Microchip Technology Inc.

AK4954 Codec Driver Library

This topic describes the AK4954 Codec Driver Library.

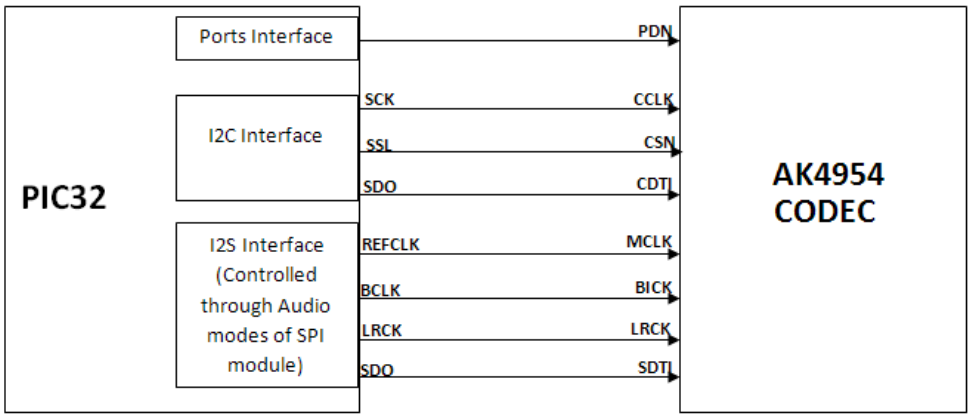
Introduction

This library provides an interface to manage the AK4954 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4954 module is 16/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK4954 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4954 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4954 Codec supports the following features:

- Audio Interface Format: MSB first
- ADC: 24-bit MSB justified, 16/24-bit I2S
- DAC: 24-bit MSB justified, 1-6bit LSB justified, 24-bit LSB justified, 16/24-bit I2S
- Sampling Frequency Range: 8 kHz to 192 kHz
- Digital Volume Control: +12dB ~ .115dB, 0.5dB Step
- SoftMute: On and Off
- Master Clock Frequencies: 32 fs/64 fs/128 fs/256 fs

Using the Library

This topic describes the basic architecture of the AK4954 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ak4954.h`

The interface to the AK4954 Codec Driver library is defined in the `drv_ak4954.h` header file. Any C language source (`.c`) file that uses the AK4954 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

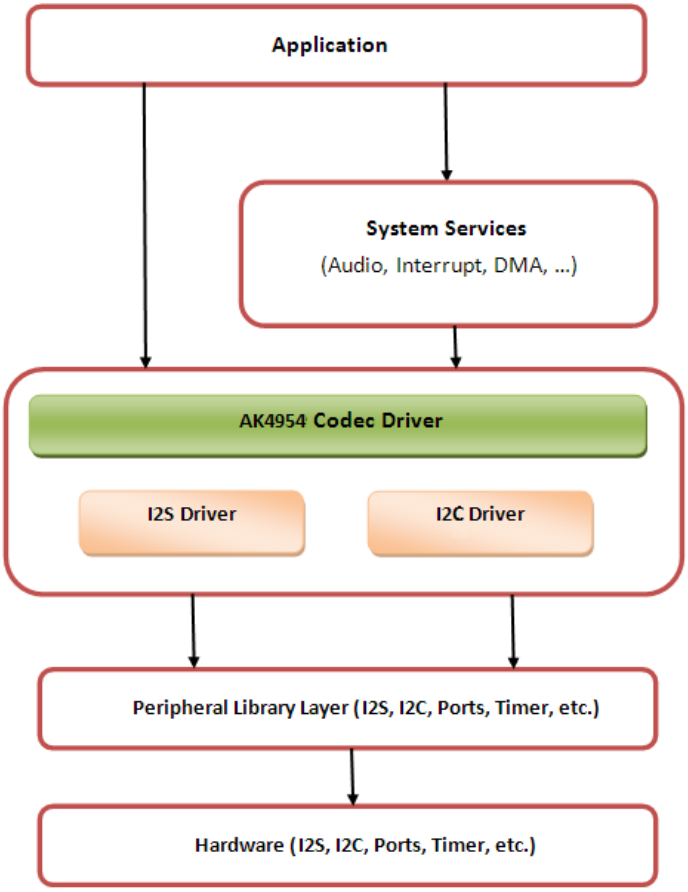
Abstraction Model

This library provides a low-level abstraction of the AK4954 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4954 Codec Driver is positioned in the MPLAB Harmony framework. The AK4954 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4954 module.

AK4954 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4954 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4954 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4954 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Status Functions	Provides status functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4954 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4954 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4954_INIT or by using Initialization Overrides) that are supported by the specific AK4954 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK4954_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4954_Deinitialize, DRV_ AK4954_Status and DRV_I2S_Tasks.

Implementations

The AK4954 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
DRV_AK4954_INIT drvak4954Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4954_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4954_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4954_VOLUME,
    .queueSizeTransmit = DRV_AK4954_TRANSMIT_QUEUE_SIZE,
};

// Initialize the I2C driver
   DRV_I2C0_Initialize();

// Initialize the I2S driver. The I2S module index should be same as the one used in initializing
// the I2S driver.
   sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)&drvI2S0InitData);

// Initialize the Codec driver
   sysObj.drvak4954Codec0 = DRV_AK4954_Initialize(DRV_AK4954_INDEX_0, (SYS_MODULE_INIT
*)&drvak4954Codec0InitData);

if (SYS_MODULE_OBJ_INVALID == AK4954DevObject)
{
// Handle error
}
```

Task Routine

The DRV_AK4954_Tasks will be called from the System Task Service.

Client Access

For the application to start using an instance of the module, it must call the DRV_AK4954_Open function. The DRV_AK4954_Open provides a driver handle to the AK4954 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4954_Deinitialize, the application must call the DRV_AK4954_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4954 Codec.

The following AK4954 Codec specific control command functions are provided:

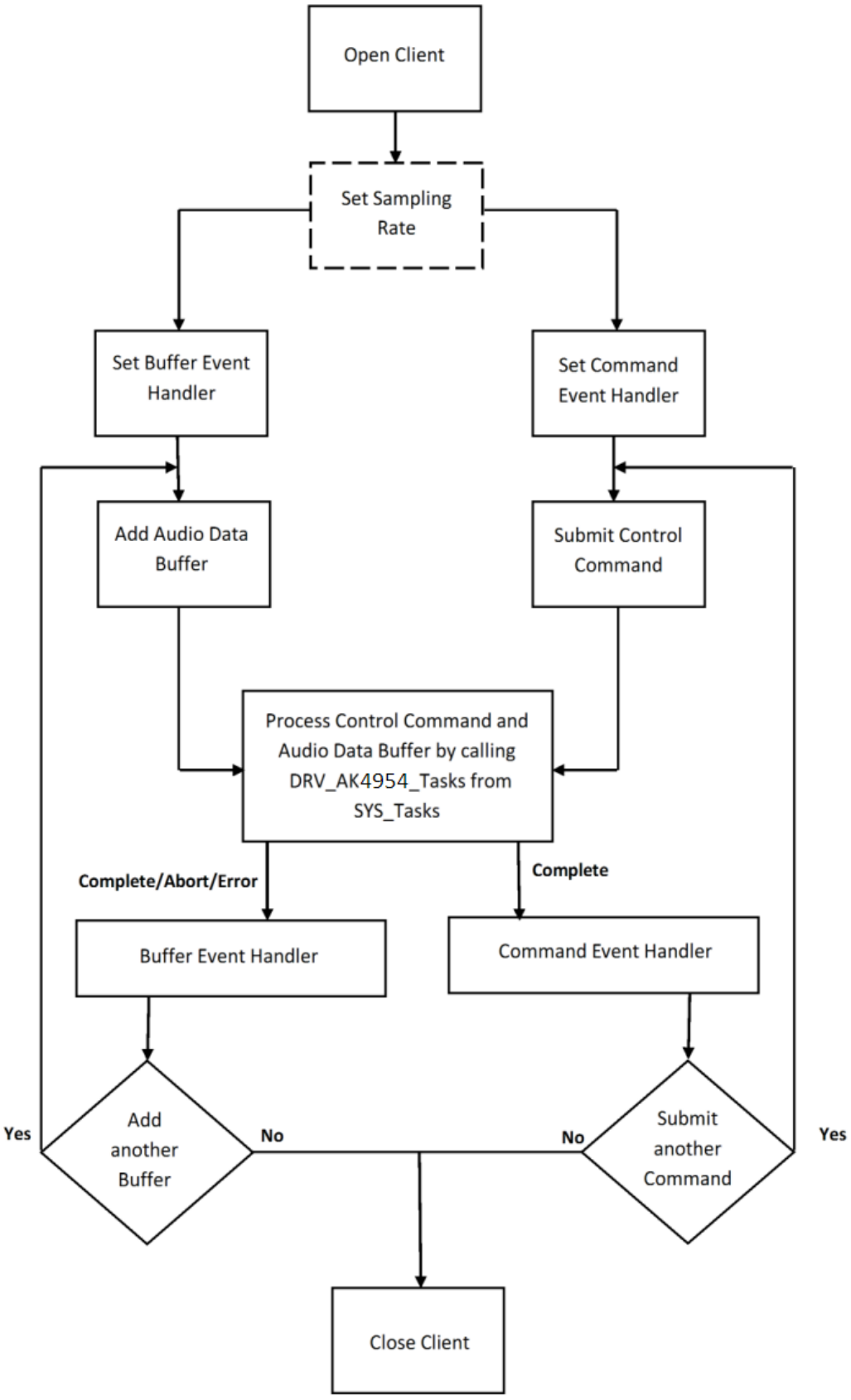
- DRV_AK4954_SamplingRateSet
- DRV_AK4954_SamplingRateGet
- DRV_AK4954_VolumeSet
- DRV_AK4954_VolumeGet
- DRV_AK4954_MuteOn
- DRV_AK4954_MuteOff
- DRV_AK4954_IntExtMicSet
- DRV_AK4954_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK4954 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_AK4954_BufferAddWrite, DRV_AK4954_BufferAddRead, and DRV_AK4954_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK4954_BUFFER_EVENT_COMPLETE, DRV_AK4954_BUFFER_EVENT_ERROR, or DRV_AK4954_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Macros

	Name	Description
	DRV_AK4954_BCLK_BIT_CLK_DIVISOR	Indicates whether the initilization of the AK4954 codec should be delayed.
	DRV_AK4954_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4954_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4954_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4954_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4954_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4954_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

The configuration of the AK4954 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4954 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4954 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4954_BCLK_BIT_CLK_DIVISOR Macro

Indicates whether the initilization of the AK4954 codec should be delayed.

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_BCLK_BIT_CLK_DIVISOR
```

Description

AK4954 Delay Initialization

If the AK4954 Codec shares its RESET pin with another peripheral, such as a Bluetooth module, then this define should be true, in order to indicate the AK4954 Codec should starts its initialization only after the other peripheral has completed theirs. It is set in the MHC menu with the checkbox: "Delay driver initialization (due to shared RESET pin)"

Remarks

This needs to be set, for example, in the case where the AK4954 and the BM64 share a common PDN (power down) or RESET pin on the PIC32 Bluetooth Audio Development Kit (BTADK).

DRV_AK4954_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_CLIENTS_NUMBER DRV_AK4954_INSTANCES_NUMBER
```

Description

AK4954 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4954 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4954_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_INPUT_REFCLOCK
```

Description

AK4954 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4954_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_INSTANCES_NUMBER
```

Description

AK4954 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4954 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4954_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4954 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency I2S sampling frequency

Supported MCLK to Sampling frequency Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs

Remarks

None

DRV_AK4954_MCLK_SOURCE Macro

Indicate the input clock frequency to generate the MCLK to codec.

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_MCLK_SOURCE
```

Description

AK4954 Data Interface Master Clock Speed configuration

Indicate the input clock frequency to generate the MCLK to codec.

Remarks

None.

DRV_AK4954_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_ak4954_config_template.h

C

```
#define DRV_AK4954_QUEUE_DEPTH_COMBINED
```

Description

AK4954 Driver Buffer Queue Entries

This macro defined the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_AK4954_BufferAddWrite function.

A buffer queue will contains buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all AK4954 driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

As an example, consider the case of a dynamic driver (say two instances) where instance one will queue up to three write requests and up to two read requests, and instance two will queue up to two write requests and up to six read requests, the value of this macro should be 13 (2 + 3 + 2 + 6).

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4954 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK4954 Codec Driver MHC Configuration

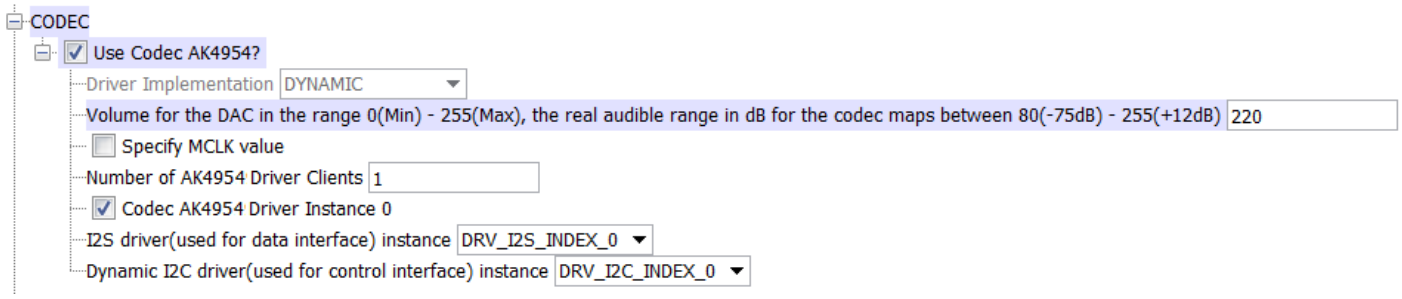


Figure 2: I2S Driver MHC Configuration

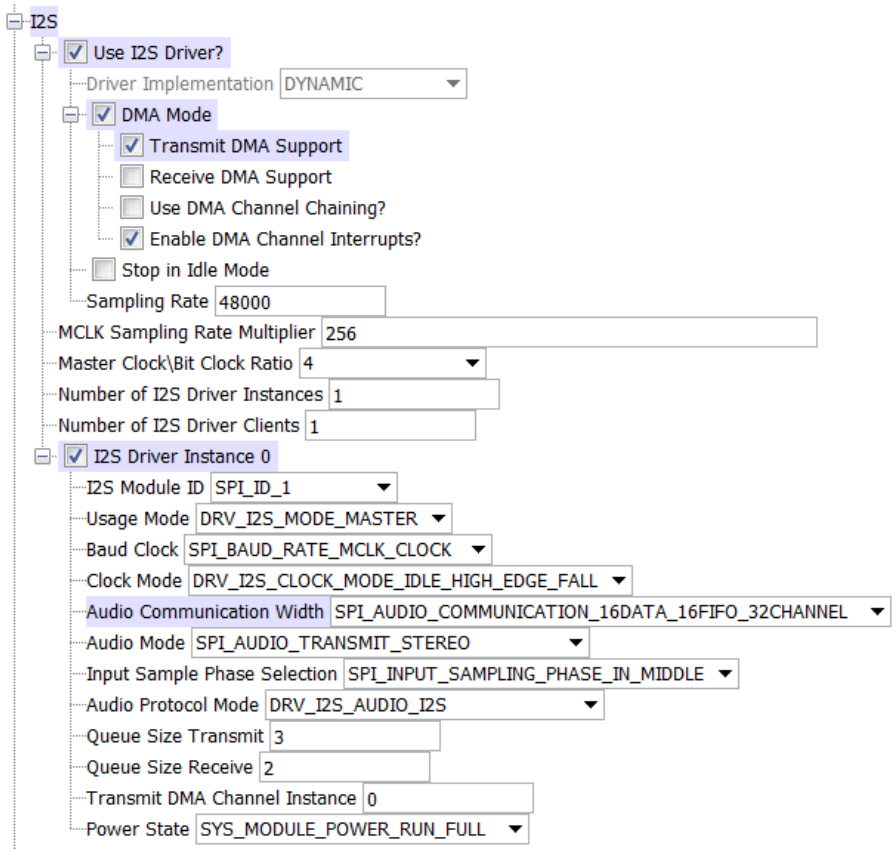
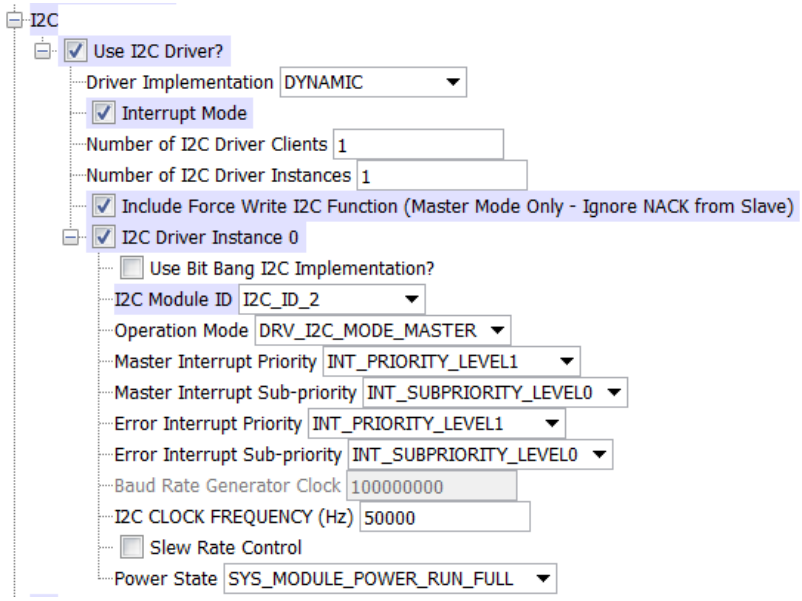


Figure 3: I2C Driver MHC Configuration



Migrating the AK4954 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK4954 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK4954 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4954 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4954`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_ak4954.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_ak4954.c	This file contains implementation of the AK4954 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AK4954 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4954_Initialize	Initializes hardware and data for the instance of the AK4954 DAC module. **Implementation:** Dynamic
	DRV_AK4954_Deinitialize	Deinitializes the specified instance of the AK4954 driver module. **Implementation:** Dynamic
	DRV_AK4954_Open	Opens the specified AK4954 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4954_Close	Closes an opened-instance of the AK4954 driver. **Implementation:** Dynamic
	DRV_AK4954_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4954_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4954_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4954_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4954_SetAudioCommunicationMode	This function provides a run time audio format configuration

b) Status Functions

	Name	Description
	DRV_AK4954_SamplingRateGet	This function gets the sampling rate set on the DAC AK4954. **Implementation:** Dynamic
	DRV_AK4954_Status	Gets the current status of the AK4954 driver module. **Implementation:** Dynamic
	DRV_AK4954_VersionGet	This function returns the version of AK4954 driver. **Implementation:** Dynamic
	DRV_AK4954_VersionStrGet	This function returns the version of AK4954 driver in string format. **Implementation:** Dynamic
	DRV_AK4954_VolumeGet	This function gets the volume for AK4954 CODEC. **Implementation:** Dynamic

c) Other Functions

	Name	Description
	DRV_AK4954_VolumeSet	This function sets the volume for AK4954 CODEC. **Implementation:** Dynamic
	DRV_AK4954_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4954_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4954_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4954_IntExtMicSet	This function sets up the codec for the X32 DB internal or the external microphone use.
	DRV_AK4954_MicSet	This function sets up the codec for the internal or the AK4954 Mic1 or Mic2 input.
	DRV_AK4954_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4954_MuteOff	This function disables AK4954 output for soft mute. **Implementation:** Dynamic
	DRV_AK4954_MuteOn	This function allows AK4954 output for soft mute on. **Implementation:** Dynamic

d) Data Types and Constants

	Name	Description
	DRV_AK4954_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4954_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4954_BUFFER_EVENT_HANDLER	Pointer to a AK4954 Driver Buffer Event handler function
	DRV_AK4954_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4954_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4954_COMMAND_EVENT_HANDLER	Pointer to a AK4954 Driver Command Event Handler Function
	DRV_AK4954_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4954_INIT	Defines the data required to initialize or reinitialize the AK4954 driver
	DRV_AK4954_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4954_MIC	This is type DRV_AK4954_MIC.
	DRV_AK4954_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.
	DRV_AK4954_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4954_COUNT	Number of valid AK4954 driver indices
	DRV_AK4954_INDEX_0	AK4954 driver index definitions
	DRV_AK4954_INDEX_1	This is macro DRV_AK4954_INDEX_1.
	DRV_AK4954_INDEX_2	This is macro DRV_AK4954_INDEX_2.
	DRV_AK4954_INDEX_3	This is macro DRV_AK4954_INDEX_3.
	DRV_AK4954_INDEX_4	This is macro DRV_AK4954_INDEX_4.
	DRV_AK4954_INDEX_5	This is macro DRV_AK4954_INDEX_5.

Description

This section describes the API functions of the AK4954 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4954_Initialize Function

Initializes hardware and data for the instance of the AK4954 DAC module.

Implementation: Dynamic

File

drv_ak4954.h

C

```
SYS_MODULE_OBJ DRV_AK4954_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4954 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4954 routine is called.

This routine should only be called once during system initialization unless DRV_AK4954_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. Also DRV_I2C_Initialize must be called before calling this function to initialize the control interface of this CODEC driver.

Example

```
DRV_AK4954_INIT                 init;
SYS_MODULE_OBJ                  objectHandle;

init->inUse                             = true;
init->status                            = SYS_STATUS_BUSY;
init->numClients                        = 0;
init->i2sDriverModuleIndex              = ak4954Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex              = ak4954Init->i2cDriverModuleIndex;
init->samplingRate                      = DRV_AK4954_AUDIO_SAMPLING_RATE;
init->audioDataFormat                   = DRV_AK4954_AUDIO_DATA_FORMAT_MACRO;
for(index=0; index < DRV_AK4954_NUMBER_OF_CHANNELS; index++)
{
    init->volume[index] = ak4954Init->volume;
}
init->isInInterruptContext              = false;

init->commandCompleteCallback = (DRV_AK4954_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;

init->mclk_multiplier = DRV_AK4954_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK4954_Initialize(DRV_AK4954_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized

init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK4954_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT *const init
);

DRV_AK4954_Deinitialize Function

Deinitializes the specified instance of the AK4954 driver module.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4954 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4954_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4954_Initialize
SYS_STATUS          status;


DRV_AK4954_Deinitialize(object);

status = DRV_AK4954_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4954_Initialize routine

Function

void DRV_AK4954_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4954_Open Function

Opens the specified AK4954 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ak4954.h

C

```
DRV_HANDLE DRV_AK4954_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK4954_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified AK4954 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

AK4954 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4954_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4954_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4954_Open(DRV_AK4954_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4954_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4954_Close Function

Closes an opened-instance of the AK4954 driver.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the AK4954 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4954_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4954_Open

DRV_AK4954_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4954_Close(DRV_Handle handle)

DRV_AK4954_Tasks Function

Maintains the driver's control and data interface state machine.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4954_Initialize

while (true)
{
    DRV_AK4954_Tasks (object);
```

```
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4954_Initialize)

Function

void DRV_AK4954_Tasks(SYS_MODULE_OBJ object);

DRV_AK4954_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4954_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4954_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4954 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4954_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

// Client registers an event handler with driver

DRV_AK4954_CommandEventHandlerSet(myAK4954Handle,
                APP_AK4954CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4954_DeEmphasisFilterSet(myAK4954Handle, DRV_AK4954_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4954CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4954_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4954_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4954_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak4954.h

C

```
void DRV_AK4954_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4954_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4954_BufferAddRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4954_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

// Client registers an event handler with driver

DRV_AK4954_BufferEventHandlerSet(myAK4954Handle,
```

```
                APP_AK4954BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4954_BufferAddRead(myAK4954handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4954_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4954BufferEventHandler(DRV_AK4954_BUFFER_EVENT event,
        DRV_AK4954_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4954_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4954_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4954_BufferEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4954_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4954_SamplingRateSet Function

This function sets the sampling rate of the media stream.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

DRV_AK4954_SamplingRateSet(myAK4954Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4954_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4954_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_ak4954.h

C

```
void DRV_AK4954_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_AK4954_SetAudioCommunicationMode

(

DRV_HANDLE handle,

const DATA_LENGTH dl,

const SAMPLE_LENGTH sl
)

b) Status Functions

DRV_AK4954_SamplingRateGet Function

This function gets the sampling rate set on the DAC AK4954.

Implementation: Dynamic

File

drv_ak4954.h

C

```
uint32_t DRV_AK4954_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

This function gets the sampling rate set on the DAC AK4954.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t baudRate;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

baudRate = DRV_AK4954_SamplingRateGet(myAK4954Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4954_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4954_Status Function

Gets the current status of the AK4954 driver module.

Implementation: Dynamic

File

drv_ak4954.h

C

```
SYS_STATUS DRV_AK4954_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4954 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4954_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4954_Initialize
SYS_STATUS          AK4954Status;

AK4954Status = DRV_AK4954_Status(object);
if (SYS_STATUS_READY == AK4954Status)
{
    // This means the driver can be opened using the
    // DRV_AK4954_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4954_Initialize routine

Function

SYS_STATUS DRV_AK4954_Status(SYS_MODULE_OBJ object)

DRV_AK4954_VersionGet Function

This function returns the version of AK4954 driver.

Implementation: Dynamic

File

drv_ak4954.h

C

```
uint32_t DRV_AK4954_VersionGet();
```

Returns

returns the version of AK4954 driver.

Description

The version number returned from the DRV_AK4954_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
    uint32_t AK4954version;
    AK4954version = DRV_AK4954_VersionGet();
```

Function

uint32_t DRV_AK4954_VersionGet(void)

DRV_AK4954_VersionStrGet Function

This function returns the version of AK4954 driver in string format.

Implementation: Dynamic

File

drv_ak4954.h

C

```
int8_t* DRV_AK4954_VersionStrGet();
```

Returns

returns a string containing the version of AK4954 driver.

Description

The DRV_AK4954_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4954 driver's version number. is the AK4954 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces.

Remarks

None.

Preconditions

None.

Example 1

"0.03a" "1.00"

Example 2

```
int8_t *AK4954string;
AK4954string = DRV_AK4954_VersionStrGet();
```

Function

int8_t* DRV_AK4954_VersionStrGet(void)

DRV_AK4954_VolumeGet Function

This function gets the volume for AK4954 CODEC.

Implementation: Dynamic

File

drv_ak4954.h

C

```
uint8_t DRV_AK4954_VolumeGet(DRV_HANDLE handle, DRV_AK4954_CHANNEL chan);
```

Returns

None.

Description

This functions gets the current volume programmed to the CODEC AK4954.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

  volume = DRV_AK4954_VolumeGet(myAK4954Handle,DRV_AK4954_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set

Function

uint8_t DRV_AK4954_VolumeGet(DRV_HANDLE handle, DRV_AK4954_CHANNEL chan)

c) Other Functions

DRV_AK4954_VolumeSet Function

This function sets the volume for AK4954 CODEC.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_VolumeSet(DRV_HANDLE handle, DRV_AK4954_CHANNEL channel, uint8_t volume);
```

Returns

None.

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

    DRV_AK4954_VolumeSet(myAK4954Handle, DRV_AK4954_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_AK4954_VolumeSet(DRV_HANDLE handle, DRV_AK4954_CHANNEL channel, uint8_t volume);

DRV_AK4954_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak4954.h

C

```
void DRV_AK4954_BufferAddRead(const DRV_HANDLE handle, DRV_AK4954_BUFFER_HANDLE * bufferHandle, void * buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4954_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4954_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4954_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4954_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4954 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4954 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 device instance and the DRV_AK4954_Status must have returned SYS_STATUS_READY.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK4954_Open call.

Parameters

Parameters	Description
handle	Handle of the AK4954 instance as return by the DRV_AK4954_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4954_BufferAddRead

(

const DRV_HANDLE handle,

DRV_AK4954_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK4954_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4954_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4954_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4954_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4954_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4954_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4954 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4954 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 device instance and the DRV_AK4954_Status must have returned SYS_STATUS_READY.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4954_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4954_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

// Client registers an event handler with driver

DRV_AK4954_BufferEventHandlerSet(myAK4954Handle,
                APP_AK4954BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4954_BufferAddWrite(myAK4954handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4954_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.
```

```
void APP_AK4954BufferEventHandler(DRV_AK4954_BUFFER_EVENT event,
        DRV_AK4954_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4954_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4954_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4954 instance as return by the DRV_AK4954_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4954_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK4954_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4954_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK4954_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4954_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4954_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4954_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4954_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4954 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4954 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every AK4954 write. The transmit and receive size must be same.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 device instance and the DRV_AK4954_Status must have returned SYS_STATUS_READY.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_AK4954_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_AK4954_BUFFER_HANDLE bufferHandle;

// myak4954Handle is the handle returned
// by the DRV_AK4954_Open function.

// Client registers an event handler with driver

DRV_AK4954_BufferEventHandlerSet(myak4954Handle,
                APP_AK4954BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4954_BufferAddWriteRead(myak4954handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_AK4954_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4954BufferEventHandler(DRV_AK4954_BUFFER_EVENT event,
        DRV_AK4954_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4954_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4954_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4954 instance as returned by the DRV_AK4954_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle

transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_AK4954_BufferAddWriteRead
(
const DRV_HANDLE handle,
DRV_AK4954_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)

DRV_AK4954_IntExtMicSet Function

This function sets up the codec for the X32 DB internal or the external microphone use.

File

drv_ak4954.h

C

```
void DRV_AK4954_IntExtMicSet(DRV_HANDLE handle, DRV_AK4954_INT_EXT_MIC micInput);
```

Returns

None

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.
DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4954_IntExtMicSet

DRV_AK4954_MicSet Function

This function sets up the codec for the internal or the AK4954 Mic1 or Mic2 input.

File

drv_ak4954.h

C

```
void DRV_AK4954_MicSet(DRV_HANDLE handle, DRV_AK4954_MIC micInput);
```

Returns

None

Description

This function sets up the codec.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4954_IntMic12Set

DRV_AK4954_MonoStereoMicSet Function

This function sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak4954.h

C

```
void DRV_AK4954_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK4954_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4954_MonoStereoMicSet(DRV_HANDLE handle);

DRV_AK4954_MuteOff Function

This function disables AK4954 output for soft mute.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4954 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.

    DRV_AK4954_MuteOff(myAK4954Handle); //AK4954 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4954_MuteOff(DRV_HANDLE handle)

DRV_AK4954_MuteOn Function

This function allows AK4954 output for soft mute on.

Implementation: Dynamic

File

drv_ak4954.h

C

```
void DRV_AK4954_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4954 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4954_Initialize routine must have been called for the specified AK4954 driver instance.

DRV_AK4954_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4954Handle is the handle returned
// by the DRV_AK4954_Open function.
```

```
DRV_AK4954_MuteOn(myAK4954Handle);  //AK4954 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4954_MuteOn(DRV_HANDLE handle);

d) Data Types and Constants

DRV_AK4954_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4954.h

C

```
typedef enum {
  DRV_AK4954_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_LSB_SDTI = 0,
  DRV_AK4954_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_16BIT_LSB_SDTI,
  DRV_AK4954_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_MSB_SDTI,
  DRV_AK4954_AUDIO_DATA_FORMAT_I2S_16BIT_24BIT,
  DRV_AK4954_AUDIO_DATA_FORMAT_32BIT_MSB_SDTO_32BIT_MSB_SDTI = 6,
  DRV_AK4954_AUDIO_DATA_FORMAT_I2S_32BIT
} DRV_AK4954_AUDIO_DATA_FORMAT;
```

Description

AK4954 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_AK4954_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4954.h

C

```
typedef enum {
  DRV_AK4954_BUFFER_EVENT_COMPLETE,
  DRV_AK4954_BUFFER_EVENT_ERROR,
  DRV_AK4954_BUFFER_EVENT_ABORT
} DRV_AK4954_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4954_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4954_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4954_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4954 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4954_BufferAddWrite() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4954_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4954_BUFFER_EVENT_HANDLER Type

Pointer to a AK4954 Driver Buffer Event handler function

File

drv_ak4954.h

C

```
typedef void (* DRV_AK4954_BUFFER_EVENT_HANDLER)(DRV_AK4954_BUFFER_EVENT event, DRV_AK4954_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4954 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4954 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4954_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4954_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4954_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4954_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4954_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4954_BUFFER_EVENT event,
                               DRV_AK4954_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4954_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4954_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4954_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4954.h

C

```
typedef uintptr_t DRV_AK4954_BUFFER_HANDLE;
```

Description

AK4954 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4954_BufferAddWrite() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_AK4954_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4954.h

C

```
typedef enum {
  DRV_AK4954_CHANNEL_LEFT,
  DRV_AK4954_CHANNEL_RIGHT,
  DRV_AK4954_CHANNEL_LEFT_RIGHT,
  DRV_AK4954_NUMBER_OF_CHANNELS
} DRV_AK4954_CHANNEL;
```

Description

AK4954 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4954_COMMAND_EVENT_HANDLER Type

Pointer to a AK4954 Driver Command Event Handler Function

File

drv_ak4954.h

C

```
typedef void (* DRV_AK4954_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4954 Driver Command Event Handler Function

This data type defines the required function signature for the AK4954 driver command event handling callback function.

A command is a control instruction to the AK4954 CODEC. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4954_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4954CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4954_DIGITAL_BLOCK_CONTROL Enumeration

Identifies Bass-Boost Control function

File

drv_ak4954.h

C

```
typedef enum {
  DRV_AK4954_RECORDING_MODE,
  DRV_AK4954_PLAYBACK_MODE,
  DRV_AK4954_RECORDING_PLAYBACK_2_MODE,
  DRV_AK4954_LOOPBACK_MODE
} DRV_AK4954_DIGITAL_BLOCK_CONTROL;
```

Members

Members	Description
DRV_AK4954_RECORDING_MODE	This is the default setting
DRV_AK4954_PLAYBACK_MODE	Min control
DRV_AK4954_RECORDING_PLAYBACK_2_MODE	Medium control
DRV_AK4954_LOOPBACK_MODE	Maximum control

Description

AK4954 Bass-Boost Control

This enumeration identifies the settings for Bass-Boost Control function.

Remarks

None.

DRV_AK4954_INIT Structure

Defines the data required to initialize or reinitialize the AK4954 driver

File

drv_ak4954.h

C

```
typedef struct {
```

```
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_AK4954_AUDIO_DATA_FORMAT audioDataFormat;
  bool delayDriverInitialization;
} DRV_AK4954_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_AK4954_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format
bool delayDriverInitialization;	true if driver initialization should be delayed due to shared RESET pin

Description

AK4954 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4954 CODEC driver.

Remarks

None.

DRV_AK4954_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak4954.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK4954_INT_EXT_MIC;
```

Description

AK4954 Mic Internal / External Input

This enumeration identifies the Mic input source.

DRV_AK4954_MIC Enumeration

File

drv_ak4954.h

C

```
typedef enum {
  MIC1 = 0,
  MIC2,
  MIC3,
  DRV_AK4954_NUMBER_OF_MIC
} DRV_AK4954_MIC;
```

Members

Members	Description
MIC1 = 0	INT_MIC
MIC2	EXT_MIC
MIC3	LINE-IN

Description

This is type DRV_AK4954_MIC.

DRV_AK4954_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono / Stereo.

File

drv_ak4954.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK4954_MONO_STEREO_MIC;
```

Description

AK4954 Mic Mono / Stereo Input

This enumeration identifies the Mic input as Mono / Stereo.

DRV_AK4954_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4954.h

C

```
#define DRV_AK4954_BUFFER_HANDLE_INVALID ((DRV_AK4954_BUFFER_HANDLE)(-1))
```

Description

AK4954 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4954_BufferAddWrite() function if the buffer add request was not successful.

Remarks

None

DRV_AK4954_COUNT Macro

Number of valid AK4954 driver indices

File

drv_ak4954.h

C

```
#define DRV_AK4954_COUNT
```

Description

AK4954 Driver Module Count

This constant identifies the maximum number of AK4954 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4954 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_AK4954_INDEX_0 Macro

AK4954 driver index definitions

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_0 0
```

Description

Driver AK4954 Module Index

These constants provide AK4954 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4954_Initialize and DRV_AK4954_Open routines to identify the driver instance in use.

DRV_AK4954_INDEX_1 Macro

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_1 1
```

Description

This is macro DRV_AK4954_INDEX_1.

DRV_AK4954_INDEX_2 Macro

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_2 2
```

Description

This is macro DRV_AK4954_INDEX_2.

DRV_AK4954_INDEX_3 Macro

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_3 3
```

Description

This is macro DRV_AK4954_INDEX_3.

DRV_AK4954_INDEX_4 Macro

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_4 4
```

Description

This is macro DRV_AK4954_INDEX_4.

DRV_AK4954_INDEX_5 Macro

File

drv_ak4954.h

C

```
#define DRV_AK4954_INDEX_5 5
```

Description

This is macro DRV_AK4954_INDEX_5.

Files

Files

Name	Description
drv_ak4954.h	AK4954 CODEC Driver Interface header file
drv_ak4954_config_template.h	AK4954 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4954Codec Driver Library.

drv_ak4954.h

AK4954 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4954_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4954_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4954_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4954_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4954_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4954_MIC	This is type DRV_AK4954_MIC.
	DRV_AK4954_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Functions

	Name	Description
	DRV_AK4954_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4954_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4954_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4954_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4954_Close	Closes an opened-instance of the AK4954 driver. **Implementation:** Dynamic
	DRV_AK4954_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4954_Deinitialize	Deinitializes the specified instance of the AK4954 driver module. **Implementation:** Dynamic
	DRV_AK4954_Initialize	Initializes hardware and data for the instance of the AK4954 DAC module. **Implementation:** Dynamic
	DRV_AK4954_IntExtMicSet	This function sets up the codec for the X32 DB internal or the external microphone use.
	DRV_AK4954_MicSet	This function sets up the codec for the internal or the AK4954 Mic1 or Mic2 input.
	DRV_AK4954_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.

	DRV_AK4954_MuteOff	This function disables AK4954 output for soft mute. **Implementation:** Dynamic
	DRV_AK4954_MuteOn	This function allows AK4954 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4954_Open	Opens the specified AK4954 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4954_SamplingRateGet	This function gets the sampling rate set on the DAC AK4954. **Implementation:** Dynamic
	DRV_AK4954_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4954_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK4954_Status	Gets the current status of the AK4954 driver module. **Implementation:** Dynamic
	DRV_AK4954_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4954_VersionGet	This function returns the version of AK4954 driver. **Implementation:** Dynamic
	DRV_AK4954_VersionStrGet	This function returns the version of AK4954 driver in string format. **Implementation:** Dynamic
	DRV_AK4954_VolumeGet	This function gets the volume for AK4954 CODEC. **Implementation:** Dynamic
	DRV_AK4954_VolumeSet	This function sets the volume for AK4954 CODEC. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_AK4954_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4954_COUNT	Number of valid AK4954 driver indices
	DRV_AK4954_INDEX_0	AK4954 driver index definitions
	DRV_AK4954_INDEX_1	This is macro DRV_AK4954_INDEX_1.
	DRV_AK4954_INDEX_2	This is macro DRV_AK4954_INDEX_2.
	DRV_AK4954_INDEX_3	This is macro DRV_AK4954_INDEX_3.
	DRV_AK4954_INDEX_4	This is macro DRV_AK4954_INDEX_4.
	DRV_AK4954_INDEX_5	This is macro DRV_AK4954_INDEX_5.

Structures

	Name	Description
	DRV_AK4954_INIT	Defines the data required to initialize or reinitialize the AK4954 driver

Types

	Name	Description
	DRV_AK4954_BUFFER_EVENT_HANDLER	Pointer to a AK4954 Driver Buffer Event handler function
	DRV_AK4954_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4954_COMMAND_EVENT_HANDLER	Pointer to a AK4954 Driver Command Event Handler Function

Description

AK4954 CODEC Driver Interface

The AK4954 CODEC device driver interface provides a simple interface to manage the AK4954 106dB 192kHz 24-Bit DAC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4954 CODEC device driver.

File Name

drv_AK4954.h

Company

Microchip Technology Inc.

drv_ak4954_config_template.h

AK4954 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4954_BCLK_BIT_CLK_DIVISOR	Indicates whether the initilization of the AK4954 codec should be delayed.
	DRV_AK4954_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4954_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4954_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4954_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4954_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4954_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

AK4954 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4954_config_template.h

Company

Microchip Technology Inc.

AK7755 Codec Driver Library

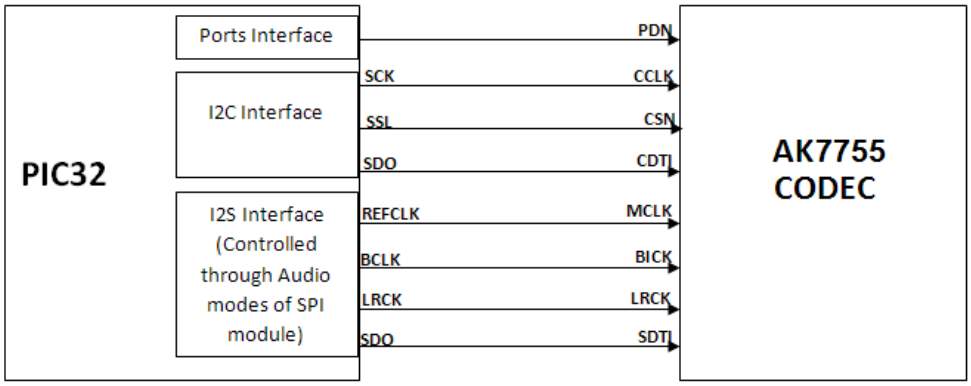
This topic describes the AK7755 Codec Driver Library.

Introduction

This library provides an interface to manage the AK7755 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK7755 module is 16/20/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK7755 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output. A typical interface of the AK7755 Codec to a Microchip PIC32 device is provided in the following diagram:



Features

The AK7755 Codec supports the following features:

- Two Digital Interfaces (I/F1, I/F2):
- 4-channel/6-channel Digital Signal Input Port: MSB justified 24-bit, LSB justified 24/20/16-bit, I2S
- Short/Long Frame
- 24-bit linear, 8-bit A-law, 8-bit µ-law

- TDM 256 fs (8-channel) MSB Justified and I2S Formats
- SoftMute: On and Off
- Stereo 24-bit ADC:
- Sampling Frequency: fs = 8 kHz ~96 kHz
- ADC Characteristics S/(N+D): 91 dB, DR, S/N: 102 dB
- Two-Channel Analog Input Selector (Differential, Single-ended Input)
- Channel Independent Microphone Analog Gain Amplifier (0 ~18 dB (2 dB Step), 18 ~36 dB (3 dB Step))
- Analog DRC (Dynamic Range Control)
- Channel Independent Digital Volume (24 ~-103 dB, 0.5 dB Step Mute)
- Digital HPF for DC Offset Cancelling
- Mono 24-bit ADC:
- Sampling Frequency: 8 kHz ~ 96 kHz
- ADC Characteristics S/(N+D): 90 dB; DR, S/N: 100 dB
- Line Amplifier: 21 dB ~ -21 dB, 3 dB Step
- Digital Volume (24 dB ~ -103 dB, 0.5 dB step, Mute)
- Digital HPF for DC Offset Cancelling
- Stereo 24-bit DAC:
- Sampling Frequency: fs = 8 kHz ~ 96 kHz
- Digital Volume (12 dB ~ -115 dB, 0.5 step, Mute)
- Digital De-emphasis Filter (tc = 50/15 µs, fs = 32 kHz, 44.1 kHz, 48 kHz)
- Master Clock: 2560 fs (internally generated by PLL from 32, 48, 64, 128, 256 and 384 fs clock)

Using the Library

This topic describes the basic architecture of the AK7755 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_AK7755.h`

The interface to the AK7755 Codec Driver library is defined in the `drv_AK7755.h` header file. Any C language source (`.c`) file that uses the AK7755 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

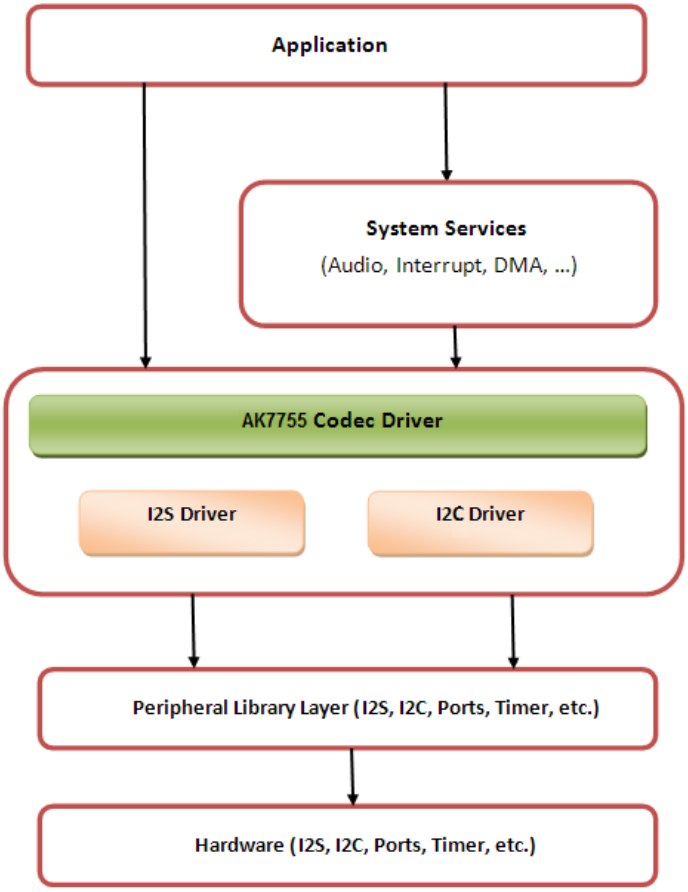
Abstraction Model

This library provides a low-level abstraction of the AK7755 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK7755 Codec Driver is positioned in the MPLAB Harmony framework. The AK7755 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK7755 module.

AK7755 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK7755 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK7755 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK7755 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Status Functions	Provides status functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK7755 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK7755 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK7755_INIT or by using Initialization Overrides) that are supported by the specific AK7755 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK7755_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK7755_Deinitialize, DRV_ AK7755_Status and DRV_I2S_Tasks.

Implementations

The AK7755 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
DRV_AK7755_INIT drvak7755Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK7755_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK7755_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK7755_VOLUME,
    .queueSizeTransmit = DRV_AK7755_TRANSMIT_QUEUE_SIZE,
};

// Initialize the I2C driver
   DRV_I2C0_Initialize();

// Initialize the I2S driver. The I2S module index should be same as the one used in initializing
// the I2S driver.
   sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)&drvI2S0InitData);

// Initialize the Codec driver
   sysObj.drvak7755Codec0 = DRV_AK7755_Initialize(DRV_AK7755_INDEX_0, (SYS_MODULE_INIT
*)&drvak7755Codec0InitData);

if (SYS_MODULE_OBJ_INVALID == AK7755DevObject)
{
// Handle error
}
```

Task Routine

The DRV_AK7755_Tasks will be called from the System Task Service.

Client Access

For the application to start using an instance of the module, it must call the DRV_AK7755_Open function. The DRV_AK7755_Open provides a driver handle to the AK7755 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK7755_Deinitialize, the application must call the DRV_AK7755_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the AK7755 Codec.

The following AK7755 Codec specific control command functions are provided:

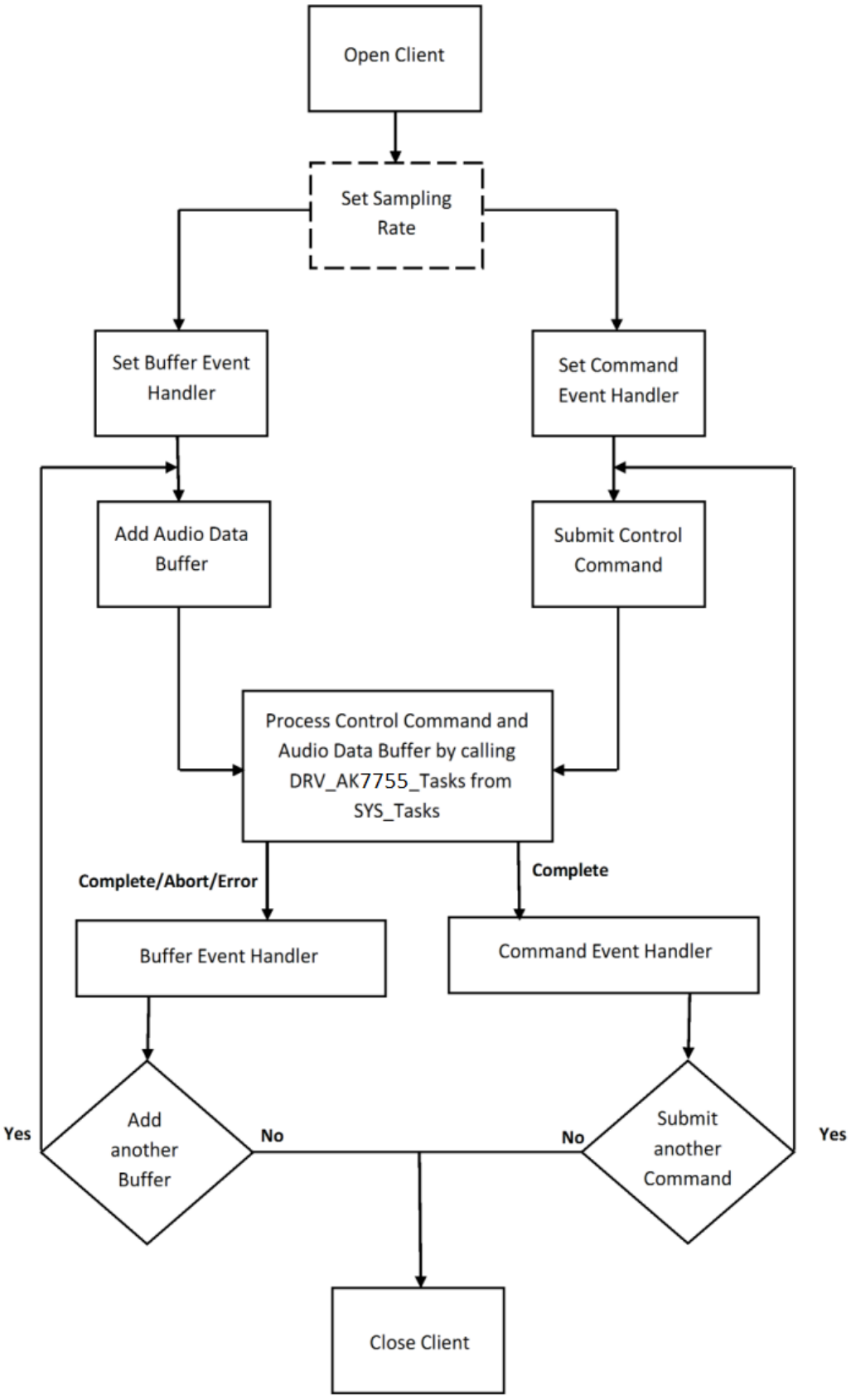
- DRV_AK7755_SamplingRateSet
- DRV_AK7755_SamplingRateGet
- DRV_AK7755_VolumeSet
- DRV_AK7755_VolumeGet
- DRV_AK7755_MuteOn
- DRV_AK7755_MuteOff
- DRV_AK7755_IntExtMicSet
- DRV_AK7755_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK7755 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_AK7755_BufferAddWrite, DRV_AK7755_BufferAddRead, and DRV_AK7755_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK7755_BUFFER_EVENT_COMPLETE, DRV_AK7755_BUFFER_EVENT_ERROR, or DRV_AK7755_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Macros

	Name	Description
	DRV_AK7755_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK7755_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to the codec.
	DRV_AK7755_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_MCLK_SOURCE	Indicates the input clock frequency to generate the MCLK to the codec.

Description

The configuration of the AK7755 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK7755 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK7755 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK7755_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_BCLK_BIT_CLK_DIVISOR
```

Description

AK7755 BCLK to LRCK Ratio to Generate Audio Stream

This macro sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

The following BCLK to LRCK ratios are supported:

- 16-bit data 16-bit channel: 32 fs; therefore, the divisor would be 8
- 16-bit data 32-bit channel: 64 fs; therefore, the divisor would be 4

Remarks

None.

DRV_AK7755_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_CLIENTS_NUMBER DRV_AK7755_INSTANCES_NUMBER
```

Description

AK7755 Client Count Configuration

This macro sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK7755 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK7755_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to the codec.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_INPUT_REFCLOCK
```

Description

AK7755 Input reference clock

This macro identifies the input REFCLOCK source to generate the MCLK to the codec.

Remarks

None.

DRV_AK7755_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_INSTANCES_NUMBER
```

Description

AK7755 driver objects configuration

This macro sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK7755 Codec modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, the driver will be built statically.

Remarks

None.

DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK7755 MCLK to LRCK Ratio to Generate Audio Stream

This macro sets up the MCLK to LRCK ratio to generate the audio stream for the specified I2S sampling frequency.

The supported MCLK to sampling frequency ratios are as follows:

- 256 fs
- 384 fs
- 512 fs
- 768 fs
- 1152 fs

Remarks

None.

DRV_AK7755_MCLK_SOURCE Macro

Indicates the input clock frequency to generate the MCLK to the codec.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_MCLK_SOURCE
```

Description

AK7755 Data Interface Master Clock Speed configuration

This macro indicates the input clock frequency to generate the MCLK to the codec.

Remarks

None.

Configuring the MHC

Description

The following three figures show examples of MHC configurations for the AK7755 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK7755 Codec Driver MHC Configuration

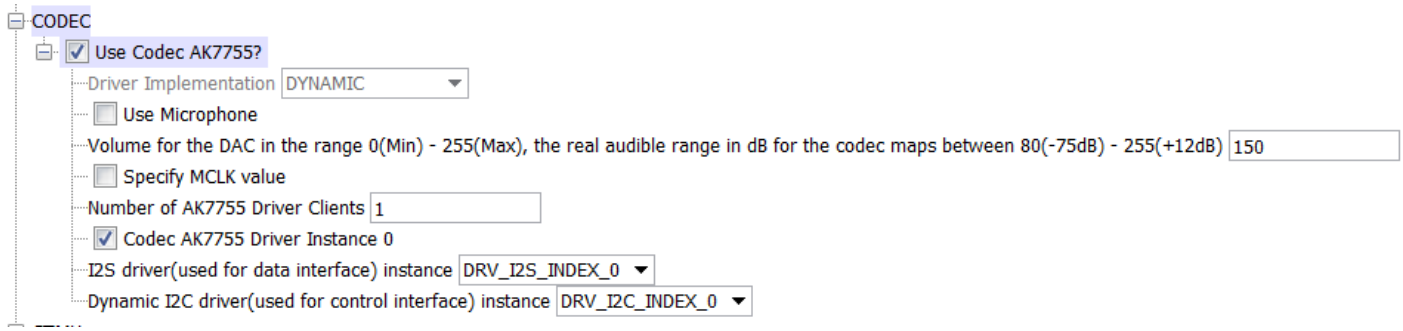


Figure 2: I2S Driver MHC Configuration

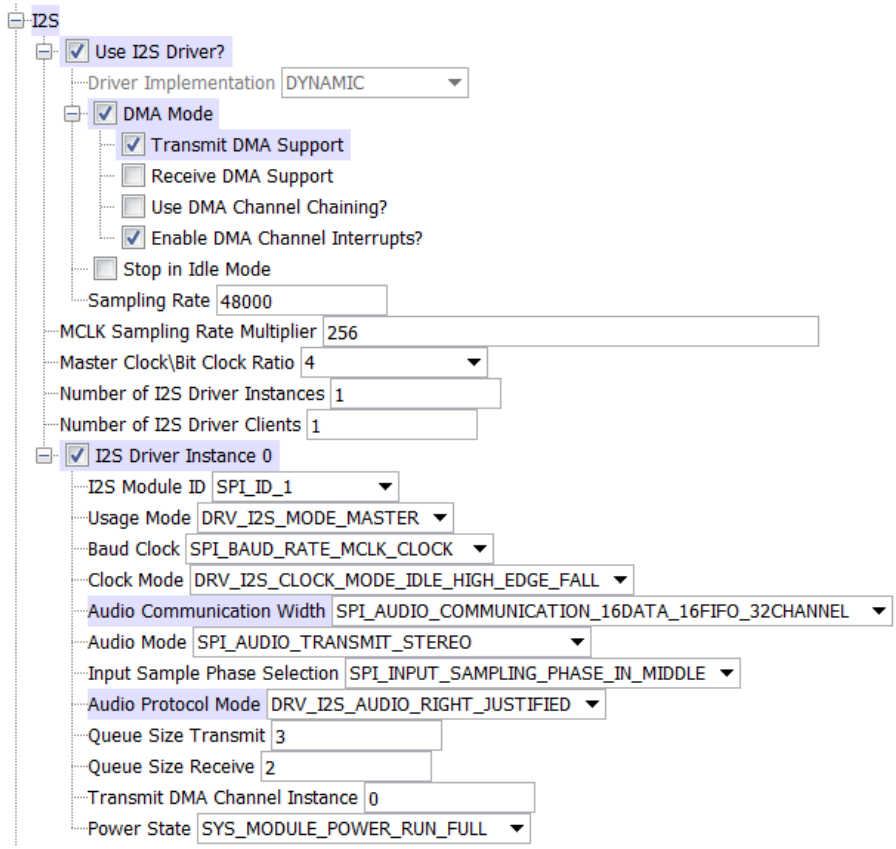
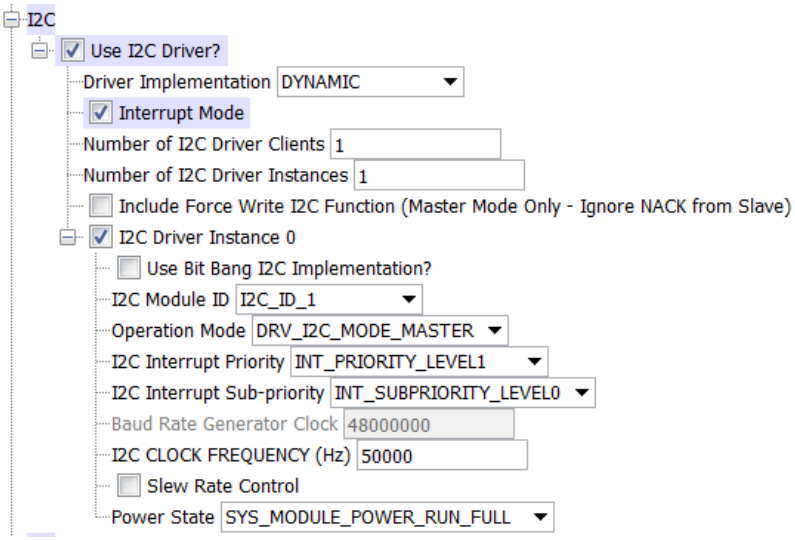


Figure 3: I2C Driver MHC Configuration



Migrating the AK7755 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK7755 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK7755 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK7755 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak7755`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak7755.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak7755.c`	This file contains implementation of the AK7755 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AK7755 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK7755_Close	Closes an opened-instance of the AK7755 Codec Driver.
	DRV_AK7755_Deinitialize	Deinitializes the specified instance of the AK7755 Codec Driver module.
	DRV_AK7755_Initialize	Initializes hardware and data for the instance of the AK7755 DAC module
	DRV_AK7755_Open	Opens the specified AK7755 Codec Driver instance and returns a handle to it
	DRV_AK7755_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK7755_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK7755_CommandEventHandlerSet	Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK7755_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK7755_SetAudioCommunicationMode	This function provides a run time audio format configuration

b) Status Functions

	Name	Description
	DRV_AK7755_SamplingRateGet	This function gets the sampling rate set on the AK7755. **Implementation:** Dynamic
	DRV_AK7755_Status	Gets the current status of the AK7755 Codec Driver module.
	DRV_AK7755_VersionGet	Returns the version of the AK7755 Codec Driver.
	DRV_AK7755_VersionStrGet	This function returns the version of AK7755 Codec Driver in string format.
	DRV_AK7755_VolumeGet	Gets the volume for the AK7755 Codec Driver.

c) Other Functions

	Name	Description
	DRV_AK7755_VolumeSet	This function sets the volume for AK7755 CODEC.
	DRV_AK7755_BufferAddRead	Schedule a non-blocking driver read operation.

	DRV_AK7755_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK7755_BufferAddWriteRead	This is function DRV_AK7755_BufferAddWriteRead.
	DRV_AK7755_IntExtMicSet	Sets up the codec for the internal or the external microphone use.
	DRV_AK7755_MonoStereoMicSet	Sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK7755_MuteOff	Disables AK7755 output for soft mute.
	DRV_AK7755_MuteOn	Allows AK7755 output for soft mute on.

d) Data Types and Constants

	Name	Description
	_DRV_AK7755_H	Include files.
	DRV_AK7755_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK7755_COUNT	Number of valid AK7755 Codec Driver indices
	DRV_AK7755_INDEX_0	AK7755 driver index definitions
	DRV_AK7755_INDEX_1	This is macro DRV_AK7755_INDEX_1.
	DRV_AK7755_INDEX_2	This is macro DRV_AK7755_INDEX_2.
	DRV_AK7755_INDEX_3	This is macro DRV_AK7755_INDEX_3.
	DRV_AK7755_INDEX_4	This is macro DRV_AK7755_INDEX_4.
	DRV_AK7755_INDEX_5	This is macro DRV_AK7755_INDEX_5.
	DRV_AK7755_BICK_FS_FORMAT	This is type DRV_AK7755_BICK_FS_FORMAT.
	DRV_AK7755_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK7755_BUFFER_EVENT_HANDLER	Pointer to a AK7755 Driver Buffer Event handler function.
	DRV_AK7755_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK7755_CHANNEL	Identifies left/right audio channel.
	DRV_AK7755_COMMAND_EVENT_HANDLER	Pointer to a AK7755 Codec Driver command event handler function.
	DRV_AK7755_DAC_INPUT_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK7755_DSP_DIN1_INPUT_FORMAT	This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.
	DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.
	DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.
	DRV_AK7755_DSP_PROGRAM	This is type DRV_AK7755_DSP_PROGRAM.
	DRV_AK7755_INIT	Defines the data required to initialize or reinitialize the AK7755 Codec Driver.
	DRV_AK7755_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK7755_LRCK_IF_FORMAT	This is type DRV_AK7755_LRCK_IF_FORMAT.
	DRV_AK7755_MONO_STEREO_MIC	Identifies the Mic input as Mono/Stereo.
	DRV_I2C_INDEX	This is macro DRV_I2C_INDEX.
	DATA_LENGTH	in bits
	SAMPLE_LENGTH	in bits

Description

This section describes the API functions of the AK7755 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK7755_Close Function

Closes an opened-instance of the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
void DRV_AK7755_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the AK7755 Codec Driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling DRV_AK7755_Open before the caller may use the driver again.

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this function is called.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK7755_Open

DRV_AK7755_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_Close(DRV_Handle handle)

DRV_AK7755_Deinitialize Function

Deinitializes the specified instance of the AK7755 Codec Driver module.

File

drv_ak7755.h

C

```
void DRV_AK7755_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the AK7755 Codec Driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

The DRV_AK7755_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK7755_Initialize
SYS_STATUS          status;

DRV_AK7755_Deinitialize(object-->);

status = DRV_AK7755_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

void DRV_AK7755_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK7755_Initialize Function

Initializes hardware and data for the instance of the AK7755 DAC module

File

drv_ak7755.h

C

```
SYS_MODULE_OBJ DRV_AK7755_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the AK7755 Codec Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This function must be called before any other AK7755 function is called.

This function should only be called once during system initialization unless DRV_AK7755_Deinitialize is called to deinitialize the driver instance. This function will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this codec driver. DRV_SPI_Initialize must be called if SPI driver is used for handling the control interface of this codec driver.

Example

```
DRV_AK7755_INIT                 init;
SYS_MODULE_OBJ                 objectHandle;

init->inUse                           = true;
init->status                          = SYS_STATUS_BUSY;
init->numClients                      = 0;
init->i2sDriverModuleIndex            = ak7755Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex            = ak7755Init->i2cDriverModuleIndex;
init->samplingRate                    = DRV_AK7755_AUDIO_SAMPLING_RATE;
init->audioDataFormat                 = DRV_AK7755_AUDIO_DATA_FORMAT_MACRO;

init->isInInterruptContext            = false;

init->commandCompleteCallback = (DRV_AK7755_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;
init->mclk_multiplier = DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK7755_Initialize(DRV_AK7755_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized

init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK7755_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT *const init

);

DRV_AK7755_Open Function

Opens the specified AK7755 Codec Driver instance and returns a handle to it

File

drv_ak7755.h

C

```
DRV_HANDLE DRV_AK7755_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK7755_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This function opens the specified AK7755 Codec Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

Only DRV_IO_INTENT_WRITE is a valid ioIntent option as AK7755 is DAC only.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK7755_Close function is called. This function will NEVER block waiting for hardware.If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK7755_Open(DRV_AK7755_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK7755_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
)

DRV_AK7755_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_ak7755.h

C

```
void DRV_AK7755_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK7755_Initialize

while (true)
{
    DRV_AK7755_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK7755_Initialize)

Function

void DRV_AK7755_Tasks(SYS_MODULE_OBJ object);

DRV_AK7755_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK7755_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK7755_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_BufferEventHandlerSet(myAK7755Handle,
                APP_AK7755BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_BufferAddWrite(myAK7755handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK7755_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK7755BufferEventHandler(DRV_AK7755_BUFFER_EVENT event,
        DRV_AK7755_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function.

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK7755_BufferEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK7755_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK7755_CommandEventHandlerSet Function

Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

File

drv_ak7755.h

C

```
void DRV_AK7755_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK7755_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK7755_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK7755 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_CommandEventHandlerSet(myAK7755Handle,
                APP_AK7755CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_DeEmphasisFilterSet(myAK7755Handle, DRV_AK7755_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK7755CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK7755_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK7755_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK7755_SamplingRateSet Function

This function sets the sampling rate of the media stream.

File

drv_ak7755.h

C

```
void DRV_AK7755_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

DRV_AK7755_SamplingRateSet(myAK7755Handle, 48000);    //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
samplingRate	Sampling frequency in Hz

Function

void DRV_AK7755_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK7755_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_ak7755.h

C

```
void DRV_AK7755_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_AK7755_Initialize routine must have been called for the specified AK7755 driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_AK7755_SetAudioCommunicationMode
(
DRV_HANDLE handle,
const DATA_LENGTH dl,
const SAMPLE_LENGTH sl
)

b) Status Functions

DRV_AK7755_SamplingRateGet Function

This function gets the sampling rate set on the AK7755.

Implementation: Dynamic

File

drv_ak7755.h

C

```
uint32_t DRV_AK7755_SamplingRateGet(DRV_HANDLE handle);
```

Description

This function gets the sampling rate set on the DAC AK7755.

Remarks

None.

Example

```
uint32_t baudRate;
```

```
// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

baudRate = DRV_AK7755_SamplingRateGet(myAK7755Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

uint32_t DRV_AK7755_SamplingRateGet(DRV_HANDLE handle)

DRV_AK7755_Status Function

Gets the current status of the AK7755 Codec Driver module.

File

drv_ak7755.h

C

```
SYS_STATUS DRV_AK7755_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized
- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the AK7755 Codec Driver module.

Remarks

A driver can be opened only when its status is SYS_STATUS_READY.

Preconditions

The DRV_AK7755_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK7755_Initialize
SYS_STATUS          AK7755Status;

AK7755Status = DRV_AK7755_Status(object);
if (SYS_STATUS_READY == AK7755Status)
{
    // This means the driver can be opened using the
    // DRV_AK7755_Open function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

SYS_STATUS DRV_AK7755_Status(SYS_MODULE_OBJ object)

DRV_AK7755_VersionGet Function

Returns the version of the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
uint32_t DRV_AK7755_VersionGet();
```

Returns

Returns the version of the AK7755 Codec Driver.

Description

The version number returned from the DRV_AK7755_VersionGet function is an unsigned integer in the following decimal format:

- * 10000 + * 100 +

Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

- For version "0.03a", return: 0 * 10000 + 3 * 100 + 0
- For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t AK7755version;
AK7755version = DRV_AK7755_VersionGet();
```

Function

uint32_t DRV_AK7755_VersionGet(void)

DRV_AK7755_VersionStrGet Function

This function returns the version of AK7755 Codec Driver in string format.

File

drv_ak7755.h

C

```
int8_t* DRV_AK7755_VersionStrGet();
```

Returns

returns a string containing the version of the AK7755 Codec Driver.

Description

The DRV_AK7755_VersionStrGet function returns a string in the format: ".[.][]" Where:

- is the AK7755 Codec Driver's version number.
- is the AK7755 Codec Driver's version number.
- is an optional "patch" or "dot" release number (which is not

included in the string if it equals "00").

- is an optional release type ("a" for alpha, "b" for beta ?

not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces. For example, "0.03a" "1.00"

Remarks

None

Preconditions

None.

Example

```
int8_t *AK7755string;
AK7755string = DRV_AK7755_VersionStrGet();
```

Function

int8_t* DRV_AK7755_VersionStrGet(void)

DRV_AK7755_VolumeGet Function

Gets the volume for the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
uint8_t DRV_AK7755_VolumeGet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel);
```

Returns

None.

Description

This functions gets the current volume programmed to the AK7755 Codec Driver.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

  volume = DRV_AK7755_VolumeGet(myAK7755Handle, DRV_AK7755_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
channel	argument indicating Left or Right or Both channel volume to be modified

Function

uint8_t DRV_AK7755_VolumeGet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel)

c) Other Functions

DRV_AK7755_VolumeSet Function

This function sets the volume for AK7755 CODEC.

File

drv_ak7755.h

C

```
void DRV_AK7755_VolumeSet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel, uint8_t volume);
```

Returns

None

Description

This functions sets the volume value from 0-255, which can attenuate from -115 dB to +12 dB. All decibels below approximately -50 dB are inaudible.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

    DRV_AK7755_VolumeSet(myAK7755Handle,DRV_AK7755_CHANNEL_LEFT, 120);    //Step 120 volume
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's Open function
channel	argument indicating Left or Right or Both channel volume to be modified
volume	Updated volume specified in the range 0-255

Function

void DRV_AK7755_VolumeSet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel, uint8_t volume);

DRV_AK7755_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddRead(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK7755_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK7755_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK7755_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK7755_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK7755 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK7755 Codec Driver instance. It should not otherwise be called directly

in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 device instance and the DRV_AK7755_Status must have returned SYS_STATUS_READY.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK7755_Open call.

Parameters

Parameters	Description
handle	Handle of the AK7755 instance as return by the DRV_AK7755_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK7755_BufferAddRead

(

const DRV_HANDLE handle,

DRV_AK7755_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK7755_BufferAddWrite Function

Schedule a non-blocking driver write operation.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddWrite(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK7755_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK7755_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK7755_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK7755_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK7755 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK7755 Codec Driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 device instance and the DRV_AK7755_Status must have returned SYS_STATUS_READY.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK7755_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_BufferEventHandlerSet(myAK7755Handle,
                APP_AK7755BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_BufferAddWrite(myAK7755handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK7755_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_AK7755BufferEventHandler(DRV_AK7755_BUFFER_EVENT event,
        DRV_AK7755_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK7755 instance as return by the DRV_AK7755_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK7755_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK7755_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK7755_BufferAddWriteRead Function

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Description

This is function DRV_AK7755_BufferAddWriteRead.

DRV_AK7755_IntExtMicSet Function

Sets up the codec for the internal or the external microphone use.

File

drv_ak7755.h

C

```
void DRV_AK7755_IntExtMicSet(DRV_HANDLE handle, DRV_AK7755_INT_EXT_MIC micInput);
```

Returns

None.

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
micInput	Internal vs. External microphone input

Function

void DRV_AK7755_IntExtMicSet(DRV_HANDLE handle);

DRV_AK7755_MonoStereoMicSet Function

Sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak7755.h

C

```
void DRV_AK7755_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK7755_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None.

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
mono_stereo_mic	Mono/Stereo microphone setup

Function

void DRV_AK7755_MonoStereoMicSet(DRV_HANDLE handle);

DRV_AK7755_MuteOff Function

Disables AK7755 output for soft mute.

File

drv_ak7755.h

C

```
void DRV_AK7755_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK7755 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

    DRV_AK7755_MuteOff(myAK7755Handle);    //AK7755 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_MuteOff(DRV_HANDLE handle)

DRV_AK7755_MuteOn Function

Allows AK7755 output for soft mute on.

File

drv_ak7755.h

C

```
void DRV_AK7755_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables AK7755 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

DRV_AK7755_MuteOn(myAK7755Handle);    //AK7755 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_MuteOn(DRV_HANDLE handle);

d) Data Types and Constants

_DRV_AK7755_H Macro

File

drv_ak7755.h

C

```
#define _DRV_AK7755_H
```

Description

Include files.

DRV_AK7755_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak7755.h

C

```
#define DRV_AK7755_BUFFER_HANDLE_INVALID ((DRV_AK7755_BUFFER_HANDLE)(-1))
```

Description

AK7755 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK7755_BufferAddWrite and the DRV_AK7755_BufferAddRead function if the buffer add request was not successful.

Remarks

None.

DRV_AK7755_COUNT Macro

Number of valid AK7755 Codec Driver indices

File

drv_ak7755.h

C

```
#define DRV_AK7755_COUNT
```

Description

AK7755 Driver Module Count

This constant identifies the maximum number of AK7755 Codec Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK7755 instances on this microcontroller.

Remarks

This value is device-specific.

DRV_AK7755_INDEX_0 Macro

AK7755 driver index definitions

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_0 0
```

Description

Driver AK7755 Module Index

These constants provide AK7755 Codec Driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK7755_Initialize and DRV_AK7755_Open functions to identify the driver instance in use.

DRV_AK7755_INDEX_1 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_1 1
```

Description

This is macro DRV_AK7755_INDEX_1.

DRV_AK7755_INDEX_2 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_2 2
```

Description

This is macro DRV_AK7755_INDEX_2.

DRV_AK7755_INDEX_3 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_3 3
```

Description

This is macro DRV_AK7755_INDEX_3.

DRV_AK7755_INDEX_4 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_4 4
```

Description

This is macro DRV_AK7755_INDEX_4.

DRV_AK7755_INDEX_5 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_5 5
```

Description

This is macro DRV_AK7755_INDEX_5.

DRV_AK7755_BICK_FS_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_BICK_64FS,
  DRV_AK7755_BICK_48FS,
  DRV_AK7755_BICK_32FS,
  DRV_AK7755_BICK_256FS
} DRV_AK7755_BICK_FS_FORMAT;
```

Description

This is type DRV_AK7755_BICK_FS_FORMAT.

DRV_AK7755_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_BUFFER_EVENT_COMPLETE,
  DRV_AK7755_BUFFER_EVENT_ERROR,
  DRV_AK7755_BUFFER_EVENT_ABORT
} DRV_AK7755_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK7755_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK7755_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK7755_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK7755 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK7755_BufferAddWrite or the DRV_AK7755_BufferAddRead function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK7755_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK7755_BUFFER_EVENT_HANDLER Type

Pointer to a AK7755 Driver Buffer Event handler function.

File

drv_ak7755.h

C

```
typedef void (* DRV_AK7755_BUFFER_EVENT_HANDLER)(DRV_AK7755_BUFFER_EVENT event, DRV_AK7755_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK7755 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK7755 Codec Driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK7755_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK7755_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK7755_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK7755_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (i.e., I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK7755_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK7755_BUFFER_EVENT event,
                               DRV_AK7755_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK7755_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak7755.h

C

```
typedef uintptr_t DRV_AK7755_BUFFER_HANDLE;
```

Description

AK7755 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK7755_BufferAddWrite or DRV_AK7755_BufferAddRead function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_AK7755_CHANNEL Enumeration

Identifies left/right audio channel.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_CHANNEL_LEFT,
  DRV_AK7755_CHANNEL_RIGHT,
  DRV_AK7755_CHANNEL_LEFT_RIGHT,
  DRV_AK7755_NUMBER_OF_CHANNELS
} DRV_AK7755_CHANNEL;
```

Description

AK7755 Audio Channel

This enumeration identifies the left/right audio channel.

Remarks

None.

DRV_AK7755_COMMAND_EVENT_HANDLER Type

Pointer to a AK7755 Codec Driver command event handler function.

File

drv_ak7755.h

C

```
typedef void (* DRV_AK7755_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK7755 Driver Command Event Handler Function

This data type defines the required function signature for the AK7755 Codec Driver command event handling callback function.

A command is a control instruction to the AK7755 Codec. For example, Mute ON/OFF, Zero Detect Enable/Disable, etc.

A client must register a pointer to a command event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK7755_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK7755CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK7755_DAC_INPUT_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DAC_INPUT_24BITMSB,
  DRV_AK7755_DAC_INPUT_24BITLSB,
```

```
  DRV_AK7755_DAC_INPUT_20BITLSB,
  DRV_AK7755_DAC_INPUT_16BITLSB
} DRV_AK7755_DAC_INPUT_FORMAT;
```

Members

Members	Description
DRV_AK7755_DAC_INPUT_20BITLSB	not supported

Description

AK7755 Audio Data Format

This enumeration identifies the Serial Audio data interface format.

DRV_AK7755_DSP_DIN1_INPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DIN1_INPUT_24BITMSB,
  DRV_AK7755_DSP_DIN1_INPUT_24BITLSB,
  DRV_AK7755_DSP_DIN1_INPUT_20BITLSB,
  DRV_AK7755_DSP_DIN1_INPUT_16BITLSB
} DRV_AK7755_DSP_DIN1_INPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.

DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DOUT1_OUTPUT_24BITMSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_24BITLSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_20BITLSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_16BITLSB
} DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.

DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DOUT4_OUTPUT_24BITMSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_24BITLSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_20BITLSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_16BITLSB
} DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.

DRV_AK7755_DSP_PROGRAM Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_ECHO_CANCELLATION,
  DRV_AK7755_DSP_REGULAR
} DRV_AK7755_DSP_PROGRAM;
```

Description

This is type DRV_AK7755_DSP_PROGRAM.

DRV_AK7755_INIT Structure

Defines the data required to initialize or reinitialize the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
} DRV_AK7755_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module (I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module (I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume

Description

AK7755 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK7755 Codec Driver.

Remarks

None.

DRV_AK7755_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak7755.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK7755_INT_EXT_MIC;
```

Description

AK7755 Mic Internal / External Input

This enumeration identifies the Mic input source.

Remarks

None.

DRV_AK7755_LRCK_IF_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_LRCK_IF_STANDARD,
  DRV_AK7755_LRCK_IF_I2S_COMPATIBLE,
  DRV_AK7755_LRCK_IF_PCM_SHORT_FRAME,
  DRV_AK7755_LRCK_IF_PCM_LONG_FRAME
} DRV_AK7755_LRCK_IF_FORMAT;
```

Description

This is type DRV_AK7755_LRCK_IF_FORMAT.

DRV_AK7755_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono/Stereo.

File

drv_ak7755.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK7755_MONO_STEREO_MIC;
```

Description

AK7755 Mic Mono/Stereo Input

This enumeration identifies the Mic input as Mono/Stereo.

Remarks

None.

DRV_I2C_INDEX Macro

File

drv_wm8904.h

C

```
#define DRV_I2C_INDEX DRV_WM8904_I2C_INSTANCES_NUMBER
```

Description

This is macro DRV_I2C_INDEX.

DATA_LENGTH Enumeration

File

drv_wm8904.h

C

```
typedef enum {
  DATA_LENGTH_16,
  DATA_LENGTH_24,
  DATA_LENGTH_32
```

```
} DATA_LENGTH;
```

Description

in bits

SAMPLE_LENGTH Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  SAMPLE_LENGTH_16,
  SAMPLE_LENGTH_32
} SAMPLE_LENGTH;
```

Description

in bits

Files

Files

Name	Description
drv_ak7755.h	AK7755 CODEC Driver Interface header file
drv_ak7755_config_template.h	AK7755 Codec Driver configuration template.

Description

This section lists the source and header files used by the AK7755Codec Driver Library.

drv_ak7755.h

AK7755 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK7755_BICK_FS_FORMAT	This is type DRV_AK7755_BICK_FS_FORMAT.
	DRV_AK7755_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK7755_CHANNEL	Identifies left/right audio channel.
	DRV_AK7755_DAC_INPUT_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK7755_DSP_DIN1_INPUT_FORMAT	This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.
	DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.
	DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.
	DRV_AK7755_DSP_PROGRAM	This is type DRV_AK7755_DSP_PROGRAM.
	DRV_AK7755_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK7755_LRCK_IF_FORMAT	This is type DRV_AK7755_LRCK_IF_FORMAT.
	DRV_AK7755_MONO_STEREO_MIC	Identifies the Mic input as Mono/Stereo.
	SAMPLE_LENGTH	in bits

Functions

	Name	Description
	DRV_AK7755_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK7755_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK7755_BufferAddWriteRead	This is function DRV_AK7755_BufferAddWriteRead.
	DRV_AK7755_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK7755_Close	Closes an opened-instance of the AK7755 Codec Driver.
	DRV_AK7755_CommandEventHandlerSet	Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

	DRV_AK7755_Deinitialize	Deinitializes the specified instance of the AK7755 Codec Driver module.
	DRV_AK7755_Initialize	Initializes hardware and data for the instance of the AK7755 DAC module
	DRV_AK7755_IntExtMicSet	Sets up the codec for the internal or the external microphone use.
	DRV_AK7755_MonoStereoMicSet	Sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK7755_MuteOff	Disables AK7755 output for soft mute.
	DRV_AK7755_MuteOn	Allows AK7755 output for soft mute on.
	DRV_AK7755_Open	Opens the specified AK7755 Codec Driver instance and returns a handle to it
	DRV_AK7755_SamplingRateGet	This function gets the sampling rate set on the AK7755. **Implementation:** Dynamic
	DRV_AK7755_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK7755_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_AK7755_Status	Gets the current status of the AK7755 Codec Driver module.
	DRV_AK7755_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK7755_VersionGet	Returns the version of the AK7755 Codec Driver.
	DRV_AK7755_VersionStrGet	This function returns the version of AK7755 Codec Driver in string format.
	DRV_AK7755_VolumeGet	Gets the volume for the AK7755 Codec Driver.
	DRV_AK7755_VolumeSet	This function sets the volume for AK7755 CODEC.

Macros

	Name	Description
	_DRV_AK7755_H	Include files.
	DRV_AK7755_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK7755_COUNT	Number of valid AK7755 Codec Driver indices
	DRV_AK7755_INDEX_0	AK7755 driver index definitions
	DRV_AK7755_INDEX_1	This is macro DRV_AK7755_INDEX_1.
	DRV_AK7755_INDEX_2	This is macro DRV_AK7755_INDEX_2.
	DRV_AK7755_INDEX_3	This is macro DRV_AK7755_INDEX_3.
	DRV_AK7755_INDEX_4	This is macro DRV_AK7755_INDEX_4.
	DRV_AK7755_INDEX_5	This is macro DRV_AK7755_INDEX_5.

Structures

	Name	Description
	DRV_AK7755_INIT	Defines the data required to initialize or reinitialize the AK7755 Codec Driver.

Types

	Name	Description
	DRV_AK7755_BUFFER_EVENT_HANDLER	Pointer to a AK7755 Driver Buffer Event handler function.
	DRV_AK7755_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK7755_COMMAND_EVENT_HANDLER	Pointer to a AK7755 Codec Driver command event handler function.

Description

AK7755 CODEC Driver Interface

The AK7755 CODEC device driver interface provides a simple interface to manage the AK7755 16/24-Bit Codec that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK7755 Codec device driver.

File Name

drv_ak7755.h

Company

Microchip Technology Inc.

drv_ak7755_config_template.h

AK7755 Codec Driver configuration template.

Macros

	Name	Description
	DRV_AK7755_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK7755_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to the codec.
	DRV_AK7755_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_MCLK_SOURCE	Indicates the input clock frequency to generate the MCLK to the codec.

Description

AK7755 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak7755_config_template.h

Company

Microchip Technology Inc.

WM8904 Codec Driver Library

This topic describes the WM8904 Codec Driver Library.

Introduction

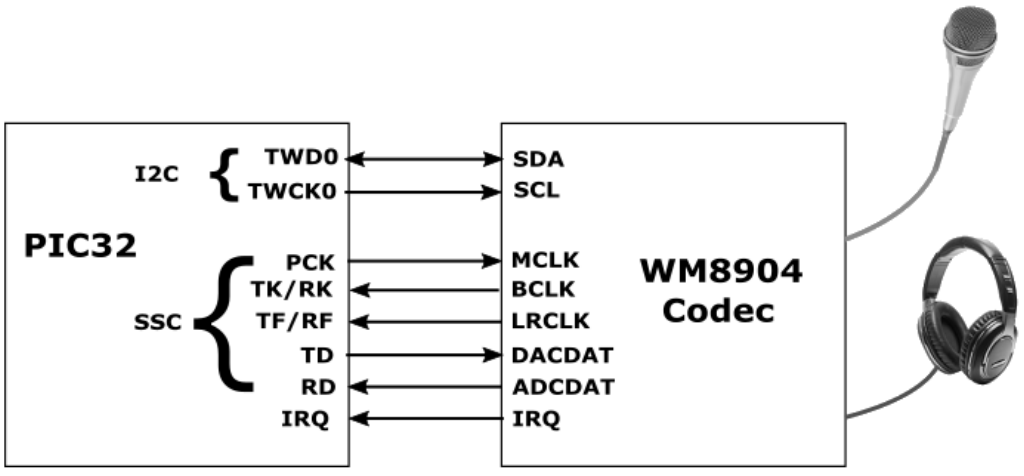
This library provides an Applications Programming Interface (API) to manage the WM8904 Codec that is serially interfaced to the I2C and I2S peripherals of a Microchip PIC32 microcontroller for the purpose of providing audio solutions.

Description

The WM8904 module is 24-bit Audio Codec from Cirrus Logic, which can operate in 16-, 20-, 24-, and 32-bit audio modes. The WM8904 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used to send commands and receive status, and the I2S interface is used for audio data output (to headphones or line-out) and input (from microphone or line-in).

The WM8904 can be configured as either an I2S clock slave (receives all clocks from the host), or I2S clock master (generates I2S clocks from a master clock input MCLK). Currently the driver only supports master mode with headphone output and (optionally) microphone input.

A typical interface of WM8904 to a Microchip PIC32 device using an I2C and SSC interface (configured as I2S), with the WM8904 set up as the I2S clock master, is provided in the following diagram:



Features

The WM8904 Codec supports the following features:

- Audio Interface Format: 16-/20-/24-/32-bit interface, LSB justified or I2S format

- Sampling Frequency Range: 8 kHz to 96 kHz
- Digital Volume Control: -71.625 to 0 dB in 192 steps
- Soft mute capability

Using the Library

This topic describes the basic architecture of the WM8904 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_WM8904.h`

The interface to the WM8904 Codec Driver library is defined in the `drv_WM8904.h` header file. Any C language source (`.c`) file that uses the WM8904 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

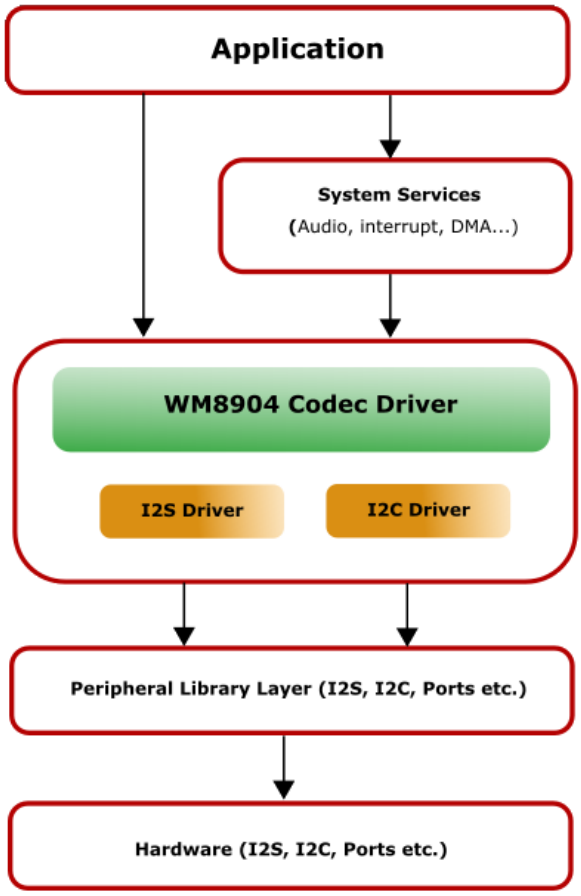
Abstraction Model

This library provides a low-level abstraction of the WM8904 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the WM8904 Codec Driver is positioned in the MPLAB Harmony framework. The WM8904 Codec Driver uses the I2C and I2S drivers for control and audio data transfers to the WM8904 module.

WM8904 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The WM8904 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced WM8904 Codec module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the WM8904 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Data Transfer Functions	Provides data transfer functions, such as Buffer Read and Write.
Settings Functions	Provides driver specific functions for settings, such as volume control and sampling rate.
Other Functions	Miscellaneous functions, such as getting the driver's version number.
Data Types and Constants	These data types and constants are required while interacting and setting up the WM8904 Codec Driver Library.

Note: All functions and constants in this section are named with the format DRV_ WM8904_xxx, where 'xxx' is a function name or constant. These names are redefined in the appropriate configuration's `system_config.h` file to the format DRV_CODEC_xxx using `#defines` so that code in the application that references the library can be written as generically as possible (e.g., by writing DRV_CODEC_Open instead of DRV_ WM8904_Open etc.). This allows the codec type to be changed in the MHC without having to modify the application's source code.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization in the `system_init.c` file, each instance of the WM8904 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_WM8904_INIT or by using Initialization Overrides) that are supported by the specific WM8904 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver
- Sampling rate
- Volume
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Determines whether or not the microphone input is enabled

The DRV_WM8904_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ WM8904_Deinitialize, DRV_ WM8904_Status and DRV_I2S_Tasks.

Implementations

The WM8904 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
SYS_STATUS status;
status = DRV_CODEC_Status(sysObjdrvCodec0);     // see if codec is done initializing
if (SYS_STATUS_READY == status)
{
    // The driver can now be opened.
   codecData->codecClient.handle = DRV_CODEC_Open
        (DRV_CODEC_INDEX_0, DRV_IO_INTENT_WRITE | DRV_IO_INTENT_EXCLUSIVE);
```

```
    if(appData.wm8904Client.handle != DRV_HANDLE_INVALID)
    {
        appData.state = APP_STATE_WM8904_SET_BUFFER_HANDLER;
    }
    else
    {
        SYS_DEBUG(0, "Find out what's wrong \r\n");
    }
}
else
{
    /* driver is not ready */
}
```

Task Routine

The DRV_WM8904_Tasks will be called from the System Task Service.

Client Access

This topic describes driver initialization and provides a code example.

Description

For the application to start using an instance of the module, it must call the DRV_WM8904_Open function. The DRV_WM8904_Open function provides a driver handle to the WM8904 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_WM8904_Deinitialize, the application must call the DRV_WM8904_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the WM8904 Codec Driver can be known by calling DRV_ WM8904_Status.

Example:

```
DRV_HANDLE handle;
SYS_STATUS wm8904Status;
wm8904Status Status = DRV_WM8904_Status(sysObjects.wm8904Status DevObject);
if (SYS_STATUS_READY == wm8904Status)
{
// The driver can now be opened.
appData.wm8904Client.handle = DRV_WM8904_Open
(DRV_WM8904_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if(appData.wm8904Client.handle != DRV_HANDLE_INVALID)
{
appData.state = APP_STATE_WM8904_SET_BUFFER_HANDLER;
}
else
{
SYS_DEBUG(0, "Find out what's wrong \r\n");
}
}
else
{
/* WM8904 Driver Is not ready */
}
```

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the WM8904 Codec.

The following WM8904 Codec specific control command functions are provided:

- DRV_WM8904_SamplingRateSet
- DRV_WM8904_SamplingRateGet
- DRV_WM8904_VolumeSet
- DRV_WM8904_VolumeGet

- DRV_WM8904_MuteOn
- DRV_WM8904_MuteOff

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the WM8904 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_WM8904_BufferAddWrite, DRV_WM8904_BufferAddRead, and DRV_WM8904_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_WM8904_BUFFER_EVENT_COMPLETE, DRV_WM8904_BUFFER_EVENT_ERROR, or DRV_WM8904_BUFFER_EVENT_ABORT events.

Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Enumerations

	Name	Description
	DRV_WM8904_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.

Macros

	Name	Description
	_DRV_WM8904_CONFIG_TEMPLATE_H	This is macro _DRV_WM8904_CONFIG_TEMPLATE_H.
	DRV_CODEC_WM8904_MODE	Specifies if codec is in Master or Slave mode.
	DRV_WM8904_BAUD_RATE	Specifies the initial baud rate for the codec.
	DRV_WM8904_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_WM8904_ENABLE_MIC_INPUT	Specifies whether to enable the microphone input.
	DRV_WM8904_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_WM8904_VOLUME	Specifies the initial volume level.

Description

The configuration of the WM8904 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the WM8904 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the WM8904 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

_DRV_WM8904_CONFIG_TEMPLATE_H Macro

File

drv_wm8904_config_template.h

C

```
#define _DRV_WM8904_CONFIG_TEMPLATE_H
```

Description

This is macro _DRV_WM8904_CONFIG_TEMPLATE_H.

DRV_CODEC_WM8904_MODE Macro

Specifies if codec is in Master or Slave mode.

File

drv_wm8904_config_template.h

C

```
#define DRV_CODEC_WM8904_MODE
```

Description

WM8904 Codec Master/Slave Mode

Indicates whether the codec is to be operating in a Master mode (generating word and bit clock as outputs) or Slave mode receiving word and bit clock as inputs).

Remarks

Only Master mode is supported at this time.

DRV_WM8904_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_wm8904.h

C

```
typedef enum {
  DATA_16_BIT_LEFT_JUSTIFIED,
  DATA_16_BIT_I2S,
  DATA_32_BIT_LEFT_JUSTIFIED,
  DATA_32_BIT_I2S
} DRV_WM8904_AUDIO_DATA_FORMAT;
```

Description

WM8904 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_WM8904_BAUD_RATE Macro

Specifies the initial baud rate for the codec.

File

drv_wm8904_config_template.h

C

```
#define DRV_WM8904_BAUD_RATE
```

Description

WM8904 Baud Rate

Sets the initial baud rate (sampling rate) for the codec. Typical values are 8000, 16000, 44100, 48000, 88200 and 96000.

Remarks

None.

DRV_WM8904_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_wm8904_config_template.h

C

```
#define DRV_WM8904_CLIENTS_NUMBER
```

Description

WM8904 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances.

Remarks

None.

DRV_WM8904_ENABLE_MIC_INPUT Macro

Specifies whether to enable the microphone input.

File

drv_wm8904_config_template.h

C

```
#define DRV_WM8904_ENABLE_MIC_INPUT
```

Description

WM8904 Microphone Enable

Indicates whether the ADC inputs for the two microphone channels (L-R) should be enabled.

Remarks

None.

DRV_WM8904_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_wm8904_config_template.h

C

```
#define DRV_WM8904_INSTANCES_NUMBER
```

Description

WM8904 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of WM8904 Codec modules that are needed by an application, namely one.

Remarks

None.

DRV_WM8904_VOLUME Macro

Specifies the initial volume level.

File

drv_wm8904_config_template.h

C

```
#define DRV_WM8904_VOLUME
```

Description

WM8904 Volume

Sets the initial volume level, in the range 0-255.

Remarks

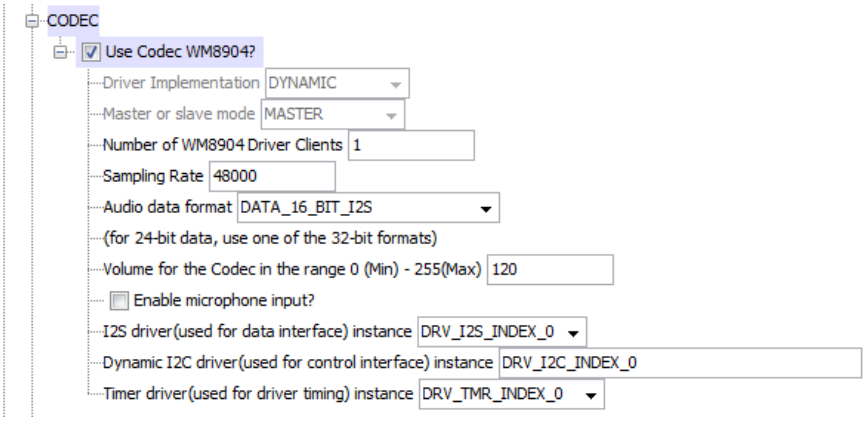
The value is mapped to an internal WM8904 volume level in the range 0-192 using a logarithmic table so the input scale appears linear (128 is half volume).

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following figure shows an example of an MHC configuration for the WM8904 Codec Driver.



Building the Library

This section lists the files that are available in the WM8904 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the WM8904 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/wm8904`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_wm8904.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_wm8904.c`	This file contains implementation of the WM8904 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The WM8904 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_WM8904_Initialize	Initializes hardware and data for the instance of the WM8904 DAC module
	DRV_WM8904_Deinitialize	Deinitializes the specified instance of the WM8904 driver module
	DRV_WM8904_Status	Gets the current status of the WM8904 driver module.
	DRV_WM8904_Tasks	Maintains the driver's control and data interface state machine.

b) Client Setup Functions

	Name	Description
	DRV_WM8904_Open	Opens the specified WM8904 driver instance and returns a handle to it
	DRV_WM8904_Close	Closes an opened-instance of the WM8904 driver
	DRV_WM8904_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_WM8904_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

c) Data Transfer Functions

	Name	Description
	DRV_WM8904_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_WM8904_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_WM8904_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic

d) Settings Functions

	Name	Description
	DRV_WM8904_MuteOff	This function disables WM8904 output for soft mute.
	DRV_WM8904_MuteOn	This function allows WM8904 output for soft mute on.
	DRV_WM8904_SamplingRateGet	This function gets the sampling rate set on the WM8904. **Implementation:** Dynamic
	DRV_WM8904_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_WM8904_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_WM8904_VolumeGet	This function gets the volume for WM8904 Codec.
	DRV_WM8904_VolumeSet	This function sets the volume for WM8904 Codec.

e) Other Functions

	Name	Description
	DRV_WM8904_VersionGet	This function returns the version of WM8904 driver
	DRV_WM8904_VersionStrGet	This function returns the version of WM8904 driver in string format.

f) Data Types and Constants

	Name	Description
	_DRV_WM8904_H	Include files.
	DRV_WM8904_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_WM8904_COUNT	Number of valid WM8904 driver indices
	DRV_WM8904_INDEX_0	WM8904 driver index definitions
	DRV_WM8904_INDEX_1	This is macro DRV_WM8904_INDEX_1.
	DRV_WM8904_INDEX_2	This is macro DRV_WM8904_INDEX_2.
	DRV_WM8904_INDEX_3	This is macro DRV_WM8904_INDEX_3.
	DRV_WM8904_INDEX_4	This is macro DRV_WM8904_INDEX_4.
	DRV_WM8904_INDEX_5	This is macro DRV_WM8904_INDEX_5.
	DRV_WM8904_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_WM8904_BUFFER_EVENT_HANDLER	Pointer to a WM8904 Driver Buffer Event handler function
	DRV_WM8904_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_WM8904_CHANNEL	Identifies Left/Right Audio channel
	DRV_WM8904_COMMAND_EVENT_HANDLER	Pointer to a WM8904 Driver Command Event Handler Function
	DRV_WM8904_INIT	Defines the data required to initialize or reinitialize the WM8904 driver

Description

This section describes the API functions of the WM8904 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_WM8904_Initialize Function

Initializes hardware and data for the instance of the WM8904 DAC module

File

drv_wm8904.h

C

```
SYS_MODULE_OBJ DRV_WM8904_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the WM8904 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other WM8904 routine is called.

This routine should only be called once during system initialization unless DRV_WM8904_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this Codec driver. DRV_I2C_Initialize must be called if SPI driver is used for handling the control interface of this Codec driver.

Example

```
DRV_WM8904_INIT                 init;
SYS_MODULE_OBJ                  objectHandle;

init->inUse                             = true;
init->status                            = SYS_STATUS_BUSY;
init->numClients                        = 0;
init->i2sDriverModuleIndex              = wm8904Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex              = wm8904Init->i2cDriverModuleIndex;
init->samplingRate                      = DRV_WM8904_AUDIO_SAMPLING_RATE;
init->audioDataFormat                   = DRV_WM8904_AUDIO_DATA_FORMAT_MACRO;

init->isInInterruptContext              = false;

init->commandCompleteCallback = (DRV_WM8904_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;
init->mclk_multiplier = DRV_WM8904_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_WM8904_Initialize(DRV_WM8904_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_WM8904_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT *const init

);

DRV_WM8904_Deinitialize Function

Deinitializes the specified instance of the WM8904 driver module

File

drv_wm8904.h

C

```
void DRV_WM8904_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the WM8904 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_WM8904_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_WM8904_Initialize
SYS_STATUS          status;


DRV_WM8904_Deinitialize(object);

status = DRV_WM8904_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_WM8904_Initialize routine

Function

void DRV_WM8904_Deinitialize(SYS_MODULE_OBJ object)

DRV_WM8904_Status Function

Gets the current status of the WM8904 driver module.

File

drv_wm8904.h

C

```
SYS_STATUS DRV_WM8904_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the WM8904 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_WM8904_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_WM8904_Initialize
SYS_STATUS          WM8904Status;

WM8904Status = DRV_WM8904_Status(object);
if (SYS_STATUS_READY == WM8904Status)
{
    // This means the driver can be opened using the
    // DRV_WM8904_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_WM8904_Initialize routine

Function

SYS_STATUS DRV_WM8904_Status(SYS_MODULE_OBJ object)

DRV_WM8904_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_wm8904.h

C

```
void DRV_WM8904_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_WM8904_Initialize

while (true)
{
    DRV_WM8904_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_WM8904_Initialize)

Function

void DRV_WM8904_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_WM8904_Open Function

Opens the specified WM8904 driver instance and returns a handle to it

File

drv_wm8904.h

C

```
DRV_HANDLE DRV_WM8904_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_WM8904_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified WM8904 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

WM8904 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_WM8904_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_WM8904_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_WM8904_Open(DRV_WM8904_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened

ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_WM8904_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
)

DRV_WM8904_Close Function

Closes an opened-instance of the WM8904 driver

File

drv_wm8904.h

C

```
void DRV_WM8904_Close(const DRV_HANDLE handle);
```

Returns

- None

Description

This routine closes an opened-instance of the WM8904 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_WM8904_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_WM8904_Open

DRV_WM8904_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_WM8904_Close(DRV_Handle handle)

DRV_WM8904_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_wm8904.h

C

```
void DRV_WM8904_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_WM8904_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_WM8904_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_WM8904_BUFFER_HANDLE bufferHandle;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

// Client registers an event handler with driver

DRV_WM8904_BufferEventHandlerSet(myWM8904Handle,
                APP_WM8904BufferEventHandler, (uintptr_t)&myAppObj);

DRV_WM8904_BufferAddWrite(myWM8904handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_WM8904_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_WM8904BufferEventHandler(DRV_WM8904_BUFFER_EVENT event,
        DRV_WM8904_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_WM8904_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_WM8904_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_WM8904_BufferEventHandlerSet
(
DRV_HANDLE handle,
const DRV_WM8904_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_WM8904_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

File

drv_wm8904.h

C

```
void DRV_WM8904_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_WM8904_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

The event handler should be set before the client performs any "WM8904 Codec Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_WM8904_BUFFER_HANDLE bufferHandle;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

// Client registers an event handler with driver

DRV_WM8904_CommandEventHandlerSet(myWM8904Handle,
                APP_WM8904CommandEventHandler, (uintptr_t)&myAppObj);

DRV_WM8904_DeEmphasisFilterSet(myWM8904Handle, DRV_WM8904_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_WM8904CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_WM8904_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_WM8904_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

c) Data Transfer Functions

DRV_WM8904_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_wm8904.h

C

```
void DRV_WM8904_BufferAddRead(const DRV_HANDLE handle, DRV_WM8904_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_WM8904_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_WM8904_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_WM8904_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_WM8904_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the WM8904 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another WM8904 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 device instance and the DRV_WM8904_Status must have returned SYS_STATUS_READY.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_WM8904_Open call.

Parameters

Parameters	Description
handle	Handle of the WM8904 instance as return by the DRV_WM8904_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

```
void DRV_WM8904_BufferAddRead
(
const       DRV_HANDLE handle,
      DRV_WM8904_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)
```

DRV_WM8904_BufferAddWrite Function

Schedule a non-blocking driver write operation.

File

drv_wm8904.h

C

```
void DRV_WM8904_BufferAddWrite(const DRV_HANDLE handle, DRV_WM8904_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_WM8904_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_WM8904_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_WM8904_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_WM8904_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the WM8904 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another WM8904 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 device instance and the DRV_WM8904_Status must have returned SYS_STATUS_READY.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_WM8904_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_WM8904_BUFFER_HANDLE bufferHandle;
```

```
// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

// Client registers an event handler with driver

DRV_WM8904_BufferEventHandlerSet(myWM8904Handle,
                APP_WM8904BufferEventHandler, (uintptr_t)&myAppObj);

DRV_WM8904_BufferAddWrite(myWM8904handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_WM8904_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_WM8904BufferEventHandler(DRV_WM8904_BUFFER_EVENT event,
        DRV_WM8904_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_WM8904_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_WM8904_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the WM8904 instance as return by the DRV_WM8904_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_WM8904_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_WM8904_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_WM8904_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_wm8904.h

C

```
void DRV_WM8904_BufferAddWriteRead(const DRV_HANDLE handle, DRV_WM8904_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_WM8904_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_WM8904_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_WM8904_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_WM8904_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the WM8904 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another WM8904 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every WM8904 write. The transmit and receive size must be same.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 device instance and the DRV_WM8904_Status must have returned SYS_STATUS_READY.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_WM8904_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_WM8904_BUFFER_HANDLE bufferHandle;

// mywm8904Handle is the handle returned
// by the DRV_WM8904_Open function.

// Client registers an event handler with driver

DRV_WM8904_BufferEventHandlerSet(mywm8904Handle,
                APP_WM8904BufferEventHandler, (uintptr_t)&myAppObj);

DRV_WM8904_BufferAddWriteRead(mywm8904handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_WM8904_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_WM8904BufferEventHandler(DRV_WM8904_BUFFER_EVENT event,
        DRV_WM8904_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
```

```
    {
        case DRV_WM8904_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_WM8904_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the WM8904 instance as returned by the DRV_WM8904_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

```
void DRV_WM8904_BufferAddWriteRead
(
const        DRV_HANDLE handle,
        DRV_WM8904_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)
```

d) Settings Functions

DRV_WM8904_MuteOff Function

This function disables WM8904 output for soft mute.

File

drv_wm8904.h

C

```
void DRV_WM8904_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables WM8904 output for soft mute.

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

    DRV_WM8904_MuteOff(myWM8904Handle); //WM8904 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_WM8904_MuteOff(DRV_HANDLE handle)

DRV_WM8904_MuteOn Function

This function allows WM8904 output for soft mute on.

File

drv_wm8904.h

C

```
void DRV_WM8904_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables WM8904 output for soft mute.

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

DRV_WM8904_MuteOn(myWM8904Handle);  //WM8904 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_WM8904_MuteOn(DRV_HANDLE handle);

DRV_WM8904_SamplingRateGet Function

This function gets the sampling rate set on the WM8904.

Implementation: Dynamic

File

drv_wm8904.h

C

```
uint32_t DRV_WM8904_SamplingRateGet(DRV_HANDLE handle);
```

Description

This function gets the sampling rate set on the DAC WM8904.

Remarks

None.

Example

```
uint32_t baudRate;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

baudRate = DRV_WM8904_SamplingRateGet(myWM8904Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_WM8904_SamplingRateGet(DRV_HANDLE handle)

DRV_WM8904_SamplingRateSet Function

This function sets the sampling rate of the media stream.

File

drv_wm8904.h

C

```
void DRV_WM8904_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

DRV_WM8904_SamplingRateSet(myWM8904Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
samplingRate	Sampling frequency in Hz

Function

void DRV_WM8904_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_WM8904_SetAudioCommunicationMode Function

This function provides a run time audio format configuration

File

drv_wm8904.h

C

```
void DRV_WM8904_SetAudioCommunicationMode(DRV_HANDLE handle, const DATA_LENGTH dl, const SAMPLE_LENGTH sl);
```

Returns

None

Description

This function sets up audio mode in I2S protocol

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
dl	Data length for I2S audio interface
sl	Left/Right Sample Length for I2S audio interface

Function

void DRV_WM8904_SetAudioCommunicationMode
(
DRV_HANDLE handle,
const DATA_LENGTH dl,
const SAMPLE_LENGTH sl
)

DRV_WM8904_VolumeGet Function

This function gets the volume for WM8904 Codec.

File

drv_wm8904.h

C

```
uint8_t DRV_WM8904_VolumeGet(DRV_HANDLE handle, DRV_WM8904_CHANNEL channel);
```

Returns

None.

Description

This functions gets the current volume programmed to the Codec WM8904.

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myWM8904Handle is the handle returned
// by the DRV_WM8904_Open function.

  volume = DRV_WM8904_VolumeGet(myWM8904Handle, DRV_WM8904_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified

Function

uint8_t DRV_WM8904_VolumeGet(DRV_HANDLE handle, DRV_WM8904_CHANNEL channel)

DRV_WM8904_VolumeSet Function

This function sets the volume for WM8904 Codec.

File

drv_wm8904.h

C

```
void DRV_WM8904_VolumeSet(DRV_HANDLE handle, DRV_WM8904_CHANNEL channel, uint8_t volume);
```

Returns

None

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_WM8904_Initialize routine must have been called for the specified WM8904 driver instance.

DRV_WM8904_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myWM8904Handle is the handle returned
```

```
// by the DRV_WM8904_Open function.

    DRV_WM8904_VolumeSet(myWM8904Handle,DRV_WM8904_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_WM8904_VolumeSet(DRV_HANDLE handle, DRV_WM8904_CHANNEL channel, uint8_t volume);

e) Other Functions

DRV_WM8904_VersionGet Function

This function returns the version of WM8904 driver

File

drv_wm8904.h

C

```
uint32_t DRV_WM8904_VersionGet();
```

Returns

returns the version of WM8904 driver.

Description

The version number returned from the DRV_WM8904_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t WM8904version;
WM8904version = DRV_WM8904_VersionGet();
```

Function

uint32_t DRV_WM8904_VersionGet(void)

DRV_WM8904_VersionStrGet Function

This function returns the version of WM8904 driver in string format.

File

drv_wm8904.h

C

```
int8_t* DRV_WM8904_VersionStrGet();
```

Returns

returns a string containing the version of WM8904 driver.

Description

The DRV_WM8904_VersionStrGet function returns a string in the format: ".[.][]" Where: is the WM8904 driver's version number. is the WM8904 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces. For example, "0.03a" "1.00"

Remarks

None

Preconditions

None.

Example

```
int8_t *WM8904string;
WM8904string = DRV_WM8904_VersionStrGet();
```

Function

int8_t* DRV_WM8904_VersionStrGet(void)

f) Data Types and Constants

_DRV_WM8904_H Macro

File

drv_wm8904.h

C

```
#define _DRV_WM8904_H
```

Description

Include files.

DRV_WM8904_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_wm8904.h

C

```
#define DRV_WM8904_BUFFER_HANDLE_INVALID ((DRV_WM8904_BUFFER_HANDLE)(-1))
```

Description

WM8904 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_WM8904_BufferAddWrite() and the DRV_WM8904_BufferAddRead() function if the buffer add request was not successful.

Remarks

None.

DRV_WM8904_COUNT Macro

Number of valid WM8904 driver indices

File

drv_wm8904.h

C

```
#define DRV_WM8904_COUNT
```

Description

WM8904 Driver Module Count

This constant identifies the maximum number of WM8904 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of WM8904 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_WM8904_INDEX_0 Macro

WM8904 driver index definitions

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_0 0
```

Description

Driver WM8904 Module Index

These constants provide WM8904 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_WM8904_Initialize and DRV_WM8904_Open routines to identify the driver instance in use.

DRV_WM8904_INDEX_1 Macro

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_1 1
```

Description

This is macro DRV_WM8904_INDEX_1.

DRV_WM8904_INDEX_2 Macro

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_2 2
```

Description

This is macro DRV_WM8904_INDEX_2.

DRV_WM8904_INDEX_3 Macro

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_3 3
```

Description

This is macro DRV_WM8904_INDEX_3.

DRV_WM8904_INDEX_4 Macro

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_4 4
```

Description

This is macro DRV_WM8904_INDEX_4.

DRV_WM8904_INDEX_5 Macro

File

drv_wm8904.h

C

```
#define DRV_WM8904_INDEX_5 5
```

Description

This is macro DRV_WM8904_INDEX_5.

DRV_WM8904_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_wm8904.h

C

```
typedef enum {
  DRV_WM8904_BUFFER_EVENT_COMPLETE,
  DRV_WM8904_BUFFER_EVENT_ERROR,
  DRV_WM8904_BUFFER_EVENT_ABORT
} DRV_WM8904_BUFFER_EVENT;
```

Members

Members	Description
DRV_WM8904_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_WM8904_BUFFER_EVENT_ERROR	Error while processing the request
DRV_WM8904_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

WM8904 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_WM8904_BufferAddWrite() or the DRV_WM8904_BufferAddRead() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_WM8904_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_WM8904_BUFFER_EVENT_HANDLER Type

Pointer to a WM8904 Driver Buffer Event handler function

File

drv_wm8904.h

C

```
typedef void (* DRV_WM8904_BUFFER_EVENT_HANDLER)(DRV_WM8904_BUFFER_EVENT event, DRV_WM8904_BUFFER_HANDLE
```

```
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

WM8904 Driver Buffer Event Handler Function

This data type defines the required function signature for the WM8904 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_WM8904_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_WM8904_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_WM8904_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_WM8904_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver(i2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_WM8904_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_WM8904_BUFFER_EVENT event,
                               DRV_WM8904_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_WM8904_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_WM8904_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_WM8904_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_wm8904.h

C

```
typedef uintptr_t DRV_WM8904_BUFFER_HANDLE;
```

Description

WM8904 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_WM8904_BufferAddWrite() or DRV_WM8904_BufferAddRead() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer.

The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_WM8904_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_wm8904.h

C

```
typedef enum {
  DRV_WM8904_CHANNEL_LEFT,
  DRV_WM8904_CHANNEL_RIGHT,
  DRV_WM8904_CHANNEL_LEFT_RIGHT,
  DRV_WM8904_NUMBER_OF_CHANNELS
} DRV_WM8904_CHANNEL;
```

Description

WM8904 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_WM8904_COMMAND_EVENT_HANDLER Type

Pointer to a WM8904 Driver Command Event Handler Function

File

drv_wm8904.h

C

```
typedef void (* DRV_WM8904_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

WM8904 Driver Command Event Handler Function

This data type defines the required function signature for the WM8904 driver command event handling callback function.

A command is a control instruction to the WM8904 Codec. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_WM8904_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_WM8904CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_WM8904_INIT Structure

Defines the data required to initialize or reinitialize the WM8904 driver

File

drv_wm8904.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_WM8904_AUDIO_DATA_FORMAT audioDataFormat;
  bool enableMicInput;
} DRV_WM8904_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of Codec
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of Codec
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_WM8904_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format
bool enableMicInput;	true if mic input path enabled

Description

WM8904 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the WM8904 Codec driver.

Remarks

None.

Files

Files

Name	Description
drv_wm8904_config_template.h	WM8904 Codec Driver Configuration Template.
drv_wm8904.h	WM8904 Codec Driver Interface header file

Description

This section lists the source and header files used by the WM8904Codec Driver Library.

drv_wm8904_config_template.h

WM8904 Codec Driver Configuration Template.

Macros

	Name	Description
	_DRV_WM8904_CONFIG_TEMPLATE_H	This is macro _DRV_WM8904_CONFIG_TEMPLATE_H.
	DRV_CODEC_WM8904_MODE	Specifies if codec is in Master or Slave mode.
	DRV_WM8904_BAUD_RATE	Specifies the initial baud rate for the codec.
	DRV_WM8904_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_WM8904_ENABLE_MIC_INPUT	Specifies whether to enable the microphone input.
	DRV_WM8904_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_WM8904_VOLUME	Specifies the initial volume level.

Description

WM8904 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_wm8904_config_template.h

Company

Microchip Technology Inc.

drv_wm8904.h

WM8904 Codec Driver Interface header file

Enumerations

	Name	Description
	DATA_LENGTH	in bits
	DRV_WM8904_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_WM8904_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_WM8904_CHANNEL	Identifies Left/Right Audio channel

Functions

	Name	Description
	DRV_WM8904_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_WM8904_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_WM8904_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_WM8904_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_WM8904_Close	Closes an opened-instance of the WM8904 driver
	DRV_WM8904_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_WM8904_Deinitialize	Deinitializes the specified instance of the WM8904 driver module
	DRV_WM8904_Initialize	Initializes hardware and data for the instance of the WM8904 DAC module
	DRV_WM8904_MuteOff	This function disables WM8904 output for soft mute.
	DRV_WM8904_MuteOn	This function allows WM8904 output for soft mute on.
	DRV_WM8904_Open	Opens the specified WM8904 driver instance and returns a handle to it
	DRV_WM8904_SamplingRateGet	This function gets the sampling rate set on the WM8904. **Implementation:** Dynamic
	DRV_WM8904_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_WM8904_SetAudioCommunicationMode	This function provides a run time audio format configuration
	DRV_WM8904_Status	Gets the current status of the WM8904 driver module.

	DRV_WM8904_Tasks	Maintains the driver's control and data interface state machine.
	DRV_WM8904_VersionGet	This function returns the version of WM8904 driver
	DRV_WM8904_VersionStrGet	This function returns the version of WM8904 driver in string format.
	DRV_WM8904_VolumeGet	This function gets the volume for WM8904 Codec.
	DRV_WM8904_VolumeSet	This function sets the volume for WM8904 Codec.

Macros

	Name	Description
	_DRV_WM8904_H	Include files.
	DRV_I2C_INDEX	This is macro DRV_I2C_INDEX.
	DRV_WM8904_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_WM8904_COUNT	Number of valid WM8904 driver indices
	DRV_WM8904_INDEX_0	WM8904 driver index definitions
	DRV_WM8904_INDEX_1	This is macro DRV_WM8904_INDEX_1.
	DRV_WM8904_INDEX_2	This is macro DRV_WM8904_INDEX_2.
	DRV_WM8904_INDEX_3	This is macro DRV_WM8904_INDEX_3.
	DRV_WM8904_INDEX_4	This is macro DRV_WM8904_INDEX_4.
	DRV_WM8904_INDEX_5	This is macro DRV_WM8904_INDEX_5.

Structures

	Name	Description
	DRV_WM8904_INIT	Defines the data required to initialize or reinitialize the WM8904 driver

Types

	Name	Description
	DRV_WM8904_BUFFER_EVENT_HANDLER	Pointer to a WM8904 Driver Buffer Event handler function
	DRV_WM8904_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_WM8904_COMMAND_EVENT_HANDLER	Pointer to a WM8904 Driver Command Event Handler Function

Description

WM8904 Codec Driver Interface

The WM8904 Codec device driver interface provides a simple interface to manage the WM8904 16/24/32-Bit Codec that can be interfaced to a Microchip microcontroller. This file provides the public interface definitions for the WM8904 Codec device driver.

File Name

drv_wm8904.h

Company

Microchip Technology Inc.

Comparator Driver Library

This section describes the Comparator Driver Library.

Introduction

The Comparator Static Driver provides a high-level interface to manage the Comparator module on the Microchip family of microcontrollers.

Description

Through MHC, this driver provides an API to initialize the Comparator module, as well as reference channels, CVREF, inputs, and interrupts.

Library Interface

Function(s)

	Name	Description
	DRV_CMP_Initialize	Initializes the Comparator instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Comparator Driver Library.

Function(s)

DRV_CMP_Initialize Function

Initializes the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_cmp.h

C

```
void DRV_CMP_Initialize();
```

Returns

None.

Description

This routine initializes the Comparator driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Comparator module ID. For example, driver instance 0 can be assigned to Comparator 2.

Remarks

This routine must be called before any other Comparator routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_CMP_Initialize(void)

CPLD XC2C64A Driver Library

This section describes the CPLD XC2C64A Driver Library.

Introduction

This library provides an interface to manage the CPLD XC2C64A devices on Microchip starter kits.

Description

A CPLD is provided on the Multimedia Expansion Board (MEB), which can be used to configure the graphics controller bus interface, SPI channel and Chip Selects used for SPI Flash, the MRF24WBOMA, and the expansion slot. The general I/O inputs are used to change the configuration, which can be done at run-time.

Specific CPLD configuration information is available in the *"Multimedia Expansion Board (MEB) User's Guide"* (DS60001160), which is available from the MEB product page: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=DM320005

Using the Library

This topic describes the basic architecture of the CPLD XC2C64A Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_xc2c64a.h`

The interface to the CPLD XC2C64A Driver Library is defined in the `drv_xc2c64a.h` header file. Any C language source (`.c`) file that uses the CPLD XC2C64A Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CPLD XC2C64A Driver.

Library Interface Section	Description
Functions	Provides CPLD XC2C64A initialization and configuration functions.

Configuring the Library

The configuration of the CPLD XC2C64A Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the CPLD XC2C64A Driver. Based on the selections made, the CPLD XC2C64A may support the selected features. These configuration settings will apply to all instances of the CPLD XC2C64A Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the CPLD XC2C64A Driver Library.

Description

This section list the files that are available in the `/src` folder of the CPLD XC2C64A Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/cpld/xc2c64a`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_xc2c64a.h`	Header file that exports the CPLD XC2C64A Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_xc2c64a.c`	Basic CPLD XC2C64A Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The CPLD XC2C64A Driver Library is not dependent on other modules.

Library Interface

a) Functions

	Name	Description
	CPLDGetDeviceConfiguration	Returns the selected device. **Implementation:** Static

	CPLDGetGraphicsConfiguration	Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller. **Implementation:** Static
	CPLDGetSPIConfiguration	Returns the selected SPI Channel. **Implementation:** Static
	CPLDInitialize	Initializes the control I/O to the CPLD and places the CPLD in a known state. **Implementation:** Static
	CPLDSetGraphicsConfiguration	Selects the PMP bus, 8 or 16-bit, interface to the graphic controller. **Implementation:** Static
	CPLDSetSPIFlashConfiguration	Selects the SPI Flash device. **Implementation:** Static
	CPLDSetWiFiConfiguration	Selects the Wi-Fi device. **Implementation:** Static
	CPLDSetZigBeeConfiguration	Selects the ZigBee/MiWi device. **Implementation:** Static

b) Data Types and Constants

	Name	Description
	CPLD_DEVICE_CONFIGURATION	CPLD device configuration.
	CPLD_GFX_CONFIGURATION	CPLD graphics controller PMP bus configuration.
	CPLD_SPI_CONFIGURATION	CPLD SPI channel selection.

Description

This section describes the API functions of the CPLD XC2C64A Driver Library.

Refer to each section for a detailed description.

a) Functions

CPLDGetDeviceConfiguration Function

Returns the selected device.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_DEVICE_CONFIGURATION CPLDGetDeviceConfiguration();
```

Returns

- CPLD_DEVICE_SPI_FLASH - SPI Flash.
- CPLD_DEVICE_WiFi - Zero G 802.11 Wi-Fi.
- CPLD_DEVICE_ZIGBEE - ZigBee/MiWi.

Description

This routine returns the selected CPLD device.

Remarks

None.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetDeviceConfiguration() != CPLD_DEVICE_SPI_FLASH)
{
    // error - not setup as default
```

```
}
```

Function

CPLD_DEVICE_CONFIGURATION CPLDGetDeviceConfiguration(void)

CPLDGetGraphicsConfiguration Function

Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_GFX_CONFIGURATION CPLDGetGraphicsConfiguration();
```

Returns

- CPLD_GFX_CONFIG_8BIT - Graphics controller is configured for 8-bit PMP data bus interface.
- CPLD_GFX_CONFIG_16BIT - Graphics controller is configured for 16-bit PMP data bus interface.

Description

This routine gets the configuration of the PMP, 8 or 16-bit, data bus interface.

Remarks

None.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetGraphicsConfiguration() != CPLD_GFX_CONFIG_8BIT)
{
    // error - not setup as default
}
```

Function

CPLD_GFX_CONFIGURATION CPLDGetGraphicsConfiguration(void)

CPLDGetSPIConfiguration Function

Returns the selected SPI Channel.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_SPI_CONFIGURATION CPLDGetSPIConfiguration();
```

Returns

- CPLD_SPI2A - SPI Channel 2A with chip select PORT G bit 9 and external interrupt 1 or 3
- CPLD_SPI3A - SPI Channel 3A with chip select PORT F bit 12 and change notice 9
- CPLD_SPI2 - SPI Channel 2 with chip select PORT G bit 9 and external interrupt 1 or 3

Description

This routine returns the selected SPI channel.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetSPIConfiguration() != CPLD_SPI2A)
{
    // error - not setup as default

}
```

Function

CPLD_SPI_CONFIGURATION CPLDGetSPIConfiguration(void)

CPLDInitialize Function

Initializes the control I/O to the CPLD and places the CPLD in a known state.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDInitialize();
```

Returns

None.

Description

This routine configures the control I/O and places the CPLD in a known state.

- Graphics Controller Bus - 8-bit PMP data interface.
- SPI Channel - SPI2/SPI2A.
- Chip Select - PORT G bit 9.
- External Interrupt 1 or 3
- Device - SPI Flash.

Remarks

None.

Preconditions

None.

Example

```
// Initialize the CPLD
CPLDInitialize();

// CPLD is configured in the default state
```

Function

void CPLDInitialize(void)

CPLDSetGraphicsConfiguration Function

Selects the PMP bus, 8 or 16-bit, interface to the graphic controller.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIGURATION configuration);
```

Returns

None.

Description

This routine sets the configuration pins on the graphics controller to select between an 8 or 16-bit data bus interface.

Remarks

The graphics controller interface configuration must be done before initializing the graphics controller.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting the graphics controller to a 16-bit interface

```
// Initialize the CPLD
CPLDInitialize();

// configure the graphics controller for a 16-bit PMP interface.
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_16BIT);
```

Setting the graphics controller to a 8-bit interface

```
// Initialize the CPLD
CPLDInitialize();

// configure the graphics controller for a 8-bit PMP interface.
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_8BIT);
```

Parameters

Parameters	Description
configuration	the type of interface configuration.

Function

void CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIGURATION configuration)

CPLDSetSPIFlashConfiguration Function

Selects the SPI Flash device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetSPIFlashConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the SPI Flash device with the selected SPI channel and Chip Select.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to SPI Flash using SPI channel 2 and chip select PORT G bit 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 2 and chip select PORT G bit 9
CPLDSetSPIFlashConfiguration(CPLD_SPI2);
```

Setting CPLD to SPI Flash using SPI channel 2A and chip select PORT G bit 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 2A and chip select PORT G bit 9
CPLDSetSPIFlashConfiguration(CPLD_SPI2A);
```

Setting CPLD to SPI Flash using SPI channel 3A and chip select PORT F bit 12

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 3A and chip select PORT F bit 12
CPLDSetSPIFlashConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the SPI Flash device.

Function

void CPLDSetSPIFlashConfiguration(CPLD_SPI_CONFIGURATION configuration)

CPLDSetWiFiConfiguration Function

Selects the Wi-Fi device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetWiFiConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the Wi-Fi device with the selected SPI channel, chip select and external interrupt or change notice.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to Wi-Fi using SPI channel 2, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 2, chip select PORT G bit 9 and external interrupt 3
CPLDSetWiFiConfiguration(CPLD_SPI2);
```

Setting CPLD to Wi-Fi using SPI channel 2A, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 2A, chip select PORT G bit 9 and external interrupt 3
CPLDSetWiFiConfiguration(CPLD_SPI2A);
```

Setting CPLD to Wi-Fi using SPI channel 3A, chip select PORT F bit 12 and change notice 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 3A, chip select PORT F bit 12 and change notice 9
CPLDSetWiFiConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the Wi-Fi device.

Function

void CPLDSetWiFiConfiguration(CPLD_SPI_CONFIGURATION configuration)

CPLDSetZigBeeConfiguration Function

Selects the ZigBee/MiWi device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetZigBeeConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the ZigBee/MiWi device with the selected SPI channel, chip select and external interrupt or change notice.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to ZigBee/MiWi using SPI channel 2, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 2, chip select PORT G bit 9 and external interrupt 3
CPLDSetZigBeeConfiguration(CPLD_SPI2);
```

Setting CPLD to ZigBee/MiWi using SPI channel 2A, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 2A, chip select PORT G bit 9 and external interrupt 3
CPLDSetZigBeeConfiguration(CPLD_SPI2A);
```

Setting CPLD to ZigBee/MiWi using SPI channel 3A, chip select PORT F bit 12 and change notice 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 3A, chip select PORT F bit 12 and change notice 9
CPLDSetZigBeeConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the ZigBee/MiWi device.

Function

void CPLDSetZigBeeConfiguration(CPLD_SPI_CONFIGURATION configuration)

b) Data Types and Constants

CPLD_DEVICE_CONFIGURATION Enumeration

CPLD device configuration.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_DEVICE_SPI_FLASH,
  CPLD_DEVICE_WiFi,
  CPLD_DEVICE_ZIGBEE
} CPLD_DEVICE_CONFIGURATION;
```

Members

Members	Description
CPLD_DEVICE_SPI_FLASH	SPI Flash
CPLD_DEVICE_WiFi	Zero G Wi-Fi
CPLD_DEVICE_ZIGBEE	ZigBee/MiWi

Description

The CPLD can be configured to communicate to three different devices. The application may call routine, CPLDGetDeviceConfiguration, to obtain what device the CPLD is configured to communicate with.

Remarks

None.

Example

```
// select 16-bit PMP data bus
if(CPLDGetDeviceConfiguration() != CPLD_DEVICE_SPI_FLASH)
{
    // error - not default configuration
}
```

CPLD_GFX_CONFIGURATION Enumeration

CPLD graphics controller PMP bus configuration.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_GFX_CONFIG_8BIT,
  CPLD_GFX_CONFIG_16BIT
} CPLD_GFX_CONFIGURATION;
```

Members

Members	Description
CPLD_GFX_CONFIG_8BIT	Configure the Graphics Controller to use 8-bit PMP data bus

CPLD_GFX_CONFIG_16BIT	Configure the Graphics Controller to use 16-bit PMP data bus

Description

The application can select what PMP bus configuration, 8 or 16-bit data bus, when interfacing with the graphics controller.

Remarks

None.

Example

```
// select 16-bit PMP data bus
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_16BIT);
```

CPLD_SPI_CONFIGURATION Enumeration

CPLD SPI channel selection.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_SPI2A,
  CPLD_SPI3A,
  CPLD_SPI2
} CPLD_SPI_CONFIGURATION;
```

Members

Members	Description
CPLD_SPI2A	PIC32 SPI Channel 2A and chip select PORT G bit 9
CPLD_SPI3A	PIC32 SPI Channel 3A and chip select PORT F bit 12
CPLD_SPI2	PIC32 SPI Channel 2 and chip select PORT G bit 9

Description

The application can select what SPI channel will be used as the communication interface. It will also select the Chip Select use for the device.

Remarks

Only one SPI channel can be select for a device. SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series devices.

Example

```
// select SPI channel two for SPI Flash
CPLDSetSPIFlashConfiguration(CPLD_SPI2);
```

Files

Files

Name	Description
drv_xc2c64a.h	This file contains the interface definition for the CUPLD controller.

Description

This section lists the source and header files used by the SPI Flash Driver Library.

drv_xc2c64a.h

This file contains the interface definition for the CUPLD controller.

Enumerations

	Name	Description
	CPLD_DEVICE_CONFIGURATION	CPLD device configuration.
	CPLD_GFX_CONFIGURATION	CPLD graphics controller PMP bus configuration.

	CPLD_SPI_CONFIGURATION	CPLD SPI channel selection.

Functions

	Name	Description
	CPLDGetDeviceConfiguration	Returns the selected device. **Implementation:** Static
	CPLDGetGraphicsConfiguration	Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller. **Implementation:** Static
	CPLDGetSPIConfiguration	Returns the selected SPI Channel. **Implementation:** Static
	CPLDInitialize	Initializes the control I/O to the CPLD and places the CPLD in a known state. **Implementation:** Static
	CPLDSetGraphicsConfiguration	Selects the PMP bus, 8 or 16-bit, interface to the graphic controller. **Implementation:** Static
	CPLDSetSPIFlashConfiguration	Selects the SPI Flash device. **Implementation:** Static
	CPLDSetWiFiConfiguration	Selects the Wi-Fi device. **Implementation:** Static
	CPLDSetZigBeeConfiguration	Selects the ZigBee/MiWi device. **Implementation:** Static

Description

CUPLD Controller Interface File.

This library provides a low-level abstraction of the CUPLD device. It can be used to simplify low-level access to the device without the necessity of interacting directly with the communication module's registers, thus hiding differences from one serial device variant to another.

File Name

drv_xc2c64a.h

Company

Microchip Technology Inc.

CTR Driver Library

This section describes the Cycle Time Register (CTR) Driver Library.

Introduction

This library provides a low-level abstraction of the Cycle Time Register (CTR) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The CTR is a hardware block that can be used to track specific signals from subsystems to internally log corresponding system time. Subsystems can include network clock synchronization, Media Clock synchronization, USB start of frame (SoF), and so on.

Using the Library

This section describes the basic architecture of the CTR Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ctr.h`

The interface to the CTR Module Library is defined in the `drv_ctr.h` header file. Any C language source (`.c`) file that uses the CTR Driver Library should include this header.

Refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the CTR Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The CTR driver provides an interface to perform a one-time configuration of the CTR peripheral. Initialization steps include selecting the mode of operation, interrupt and trigger sources, latch configurations, and so on.

In addition, the driver allows the client to register a callback that is executed when the desired event has been triggered.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CTR module.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Other Functions	Provides driver miscellaneous functions, data transfer status function, version identification functions, and so on.
Data Types and Constants	Provides data types and macros.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the modes that are supported for your device.

Configuring the Library

The configuration of the driver is based on the file `system_config.h`.

Description

The header file contains the configuration selection for the driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the CTR Driver Library.

Description

This section list the files that are available in the `/src` folder of the CTR Driver Library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/ctr/`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ctr.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_ctr.c	Basic CTR Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The CTR Driver Library depends on the following modules:

- Clock System Service Library

Optional Dependencies

- DMA System Service Library (used when operating in DMA mode)
- Interrupt System Service Library (used when task is running in Interrupt mode)

Library Interface

This section describes the API functions of the CTR Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

Functions

	Name	Description
	DRV_CTR_Deinitialize	Deinitializes the specified instance of the CTR driver module. **Implementation:** Dynamic
	DRV_CTR_Initialize	Initializes the CTR Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_CTR_Status	Gets the current status of the CTR Driver module. **Implementation:** Dynamic

Description

DRV_CTR_Deinitialize Function

Deinitializes the specified instance of the CTR driver module.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the CTR Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_CTR_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_CTR_Initialize
SYS_STATUS          status;


DRV_CTR_Deinitialize(object);

status = DRV_CTR_Status(object);
if (SYS_STATUS_UNINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CTR_Initialize

Function

void DRV_CTR_Deinitialize(SYS_MODULE_OBJ object)

DRV_CTR_Initialize Function

Initializes the CTR Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_ctr.h

C

```
SYS_MODULE_OBJ DRV_CTR_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the CTR driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other CTR function is called.

This function should only be called once during system initialization unless DRV_CTR_Deinitialize is called to deinitialize the driver instance.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the CTR Driver. All
the CTR initialization is done in #defines mentioned, and the init structure
is initialized with corresponding #defines and then passed to initialize
function.

// *****************************************************************************
// CTR Driver Configuration Options

#define DRV_CTR_POWER_STATE             SYS_MODULE_POWER_RUN_FULL
#define DRV_CTR_MODULE_ID               CTR_ID_0
#define DRV_CTR_CLIENTS_NUMBER          1
#define DRV_CTR_INSTANCES_NUMBER        1
#define DRV_CTR_EVENT_INTERRUPT_SOURCE  INT_SOURCE_CTR1_EVENT
#define DRV_CTR_EVENT_INTERRUPT_MODE    CTR_LATCH_TRIG
```

```
#define DRV_CTR_TRIGGER_INTERRUPT_SOURCE INT_SOURCE_CTR1_TRG
#define DRV_CTR_M_0                      0x000000
#define DRV_CTR_N_0                      0x000000
#define DRV_CTR_LSB_0                    0x00
#define DRV_CTR_MODE_0                   CTR_US
#define DRV_CTR_M_1                      0x000000
#define DRV_CTR_N_1                      0x000000
#define DRV_CTR_LSB_1                    0x00
#define DRV_CTR_MODE_1                   CTR_US
#define DRV_CTR_COUNTER_SEL              CTR_CTR0_LIN
#define DRV_CTR_DIVIDER                  0
#define DRIVER_MODE                      WIFI_MODE
#define DRV_CTR_LATCH0_TRIG              CTR_WIFI_TM_1
#define DRV_CTR_LATCH1_TRIG              CTR_WIFI_TM_2
#define DRV_CTR_LATCH2_TRIG              CTR_WIFI_TM_3
#define DRV_CTR_LATCH3_TRIG              CTR_WIFI_TM_4
#define DRV_CTR_TRIGGER_SOURCE               CTR_CTR0_LIN
#define DRV_CTR_TRIGGER_PHASE                0x000

DRV_CTR_INIT   CTRInitData;
SYS_MODULE_OBJ      objectHandle;

CTRInitData.moduleInit = DRV_CTR_POWER_STATE,
CTRInitData.ctrEventInterruptSource = DRV_CTR_EVENT_INTERRUPT_SOURCE,
CTRInitData.ctrLatchEventMode = DRV_CTR_EVENT_INTERRUPT_MODE,
CTRInitData.ctrTriggerInterruptSource = DRV_CTR_TRIGGER_INTERRUPT_SOURCE,
CTRInitData.ctrCounter[0].M = DRV_CTR_M_0,
CTRInitData.ctrCounter[0].N = DRV_CTR_N_0,
CTRInitData.ctrCounter[0].LSB = DRV_CTR_LSB_0,
CTRInitData.ctrCounter[1].M = DRV_CTR_M_1,
CTRInitData.ctrCounter[1].N = DRV_CTR_N_1,
CTRInitData.ctrCounter[1].LSB = DRV_CTR_LSB_1,
CTRInitData.ctrLatch[0].ctrSel = DRV_CTR_COUNTER_SEL,
CTRInitData.ctrLatch[1].ctrSel = DRV_CTR_COUNTER_SEL,
CTRInitData.ctrLatch[2].ctrSel = DRV_CTR_COUNTER_SEL,
CTRInitData.ctrLatch[3].ctrSel = DRV_CTR_COUNTER_SEL,
CTRInitData.ctrLatch[0].trigSel = DRV_CTR_LATCH0_TRIG,
CTRInitData.ctrLatch[1].trigSel = DRV_CTR_LATCH1_TRIG,
CTRInitData.ctrLatch[2].trigSel = DRV_CTR_LATCH2_TRIG,
CTRInitData.ctrLatch[3].trigSel = DRV_CTR_LATCH3_TRIG,
CTRInitData.ctrLatch[0].divider = DRV_CTR_DIVIDER,
CTRInitData.ctrLatch[1].divider = DRV_CTR_DIVIDER,
CTRInitData.ctrLatch[2].divider = DRV_CTR_DIVIDER,
CTRInitData.ctrLatch[3].divider = DRV_CTR_DIVIDER,
CTRInitData.ctrTrigger.trigSource = DRV_CTR_TRIGGER_SOURCE,
CTRInitData.ctrTrigger.phase = DRV_CTR_TRIGGER_PHASE,
CTRInitData.drvMode = DRIVER_MODE

objectHandle = DRV_CTR_Initialize(DRV_CTR_INDEX_0,
                                  (SYS_MODULE_INIT*)CTRInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_CTR_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_CTR_Status Function

Gets the current status of the CTR Driver module.

Implementation: Dynamic

File

drv_ctr.h

C

```
SYS_STATUS DRV_CTR_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the CTR Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_CTR_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_CTR_Initialize
SYS_STATUS          CTRStatus;

CTRStatus = DRV_CTR_Status(object);
if (SYS_STATUS_ERROR == CTRStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CTR_Initialize

Function

SYS_STATUS DRV_CTR_Status(SYS_MODULE_OBJ object)

b) Other Functions

Functions

	Name	Description
	DRV_CTR_Adjust	Sets the adjust value for a given CTR counter. **Implementation:** Dynamic
	DRV_CTR_ClientStatus	Gets current client-specific status of the CTR driver. **Implementation:** Dynamic
	DRV_CTR_Close	Closes an opened-instance of the CTR driver. **Implementation:** Dynamic
	DRV_CTR_Drift	Sets the drift value for a given CTR counter. **Implementation:** Dynamic
	DRV_CTR_EventISR	Interrupt Service Routine called for the CTR event interrupt. **Implementation:** Dynamic
	DRV_CTR_Open	Opens the specified CTR driver instance and returns a handle to it. **Implementation:** Dynamic

	DRV_CTR_RegisterCallBack	Registers a callback function for the event interrupt of CTR. **Implementation:** Dynamic
	DRV_CTR_TriggerISR	Interrupt Service Routine called for the CTR Trigger interrupt. **Implementation:** Dynamic

Description

DRV_CTR_Adjust Function

Sets the adjust value for a given CTR counter.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_Adjust(DRV_HANDLE handle, CTR_LATCH_CTR_SELECT ctrSel, uint16_t adjustVal);
```

Returns

None.

Description

This function sets the adjust value for a given CTR counter.

Preconditions

The DRV_CTR_Initialize function must have been called.

DRV_CTR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle; // handle returned by open function
uint16_t adjustVal = 0xFFF;

DRV_CTR_Adjust(handle, CTR_CTR0_LIN, adjustVal);
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

void DRV_CTR_Adjust(DRV_HANDLE handle, CTR_LATCH_CTR_SELECT ctrSel,

uint16_t adjustVal);

ctrSel - CTR counter to be selected out of the 4 counters available.

adjustVal - Adjust value to be set

DRV_CTR_ClientStatus Function

Gets current client-specific status of the CTR driver.

Implementation: Dynamic

File

drv_ctr.h

C

```
DRV_CTR_CLIENT_STATUS DRV_CTR_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_CTR_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the CTR driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_CTR_Initialize function must have been called.

DRV_CTR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_CTR_Open
DRV_CTR_CLIENT_STATUS     clientStatus;

clientStatus = DRV_CTR_ClientStatus(handle);
if(DRV_CTR_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_CTR_CLIENT_STATUS DRV_CTR_ClientStatus(DRV_HANDLE handle);

DRV_CTR_Close Function

Closes an opened-instance of the CTR driver.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the CTR driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_CTR_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_CTR_Initialize function must have been called for the specified CTR driver instance.

DRV_CTR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_CTR_Open

DRV_CTR_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_CTR_Close(DRV_Handle handle);

DRV_CTR_Drift Function

Sets the drift value for a given CTR counter.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_Drift(DRV_HANDLE handle, CTR_LATCH_CTR_SELECT ctrSel, uint32_t driftVal);
```

Returns

None.

Description

This function sets the drift value for a given CTR counter.

Preconditions

The DRV_CTR_Initialize function must have been called.

DRV_CTR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle; // handle returned by open function
uint16_t driftVal = 0xFFF;

DRV_CTR_Drift(handle, CTR_CTR0_LIN, driftVal);
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

void DRV_CTR_Drift(DRV_HANDLE handle, CTR_LATCH_CTR_SELECT ctrSel,

uint16_t driftVal);

ctrSel - CTR counter to be selected out of the 4 counters available.

adjustVal - Drift value to be set

DRV_CTR_EventISR Function

Interrupt Service Routine called for the CTR event interrupt.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_EventISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the interrupt is generated for CTR event interrupt. The latch buffers are read and stored in a local buffer, and all the registered client callback functions will be called from this function. The number of latches to be read depends upon the use-case configured. For wifi, 4 latches are read, and for USBSoF and GPIO, only 1 latch is read. Number of buffers to read in each latch depends on the interrupt mode configuration. For Full, all 4 buffers needs to be read, whereas for half-full, only 2 buffers needs to be read and for every trigger, only 1 buffer is

read.

Remarks

All the handling specific for a client should be done in the respective callback functions. This function should not be modified.

Preconditions

None.

Example

This function is not called from clients/system. This function will be called when the interrupt for event is generated.

Parameters

Parameters	Description
object	The driver instance handle returned after the initialization.

Function

void DRV_CTR_EventISR(SYS_MODULE_OBJ object);

DRV_CTR_Open Function

Opens the specified CTR driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ctr.h

C

```
DRV_HANDLE DRV_CTR_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_CTR_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

Description

This function opens the specified CTR driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_CTR_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_CTR_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_CTR_Open(DRV_CTR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_CTR_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
);

DRV_CTR_RegisterCallBack Function

Registers a callback function for the event interrupt of CTR.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_RegisterCallBack(const DRV_HANDLE handle, const DRV_CTR_CALLBACK callback, const bool
oneTimeCallback, const uintptr_t context);
```

Returns

None.

Description

This function registers a client callback function for the event interrupt associated with the use-case. For Wifi usecase, Only Latch 3 interrupt will be enabled, as the last event timestamp will be filled in latch 3 for IEEE 802.11v. For USBSoF and GPIO use-cases, only one latch is needed and the interrupt will be enabled for the same latch. As per user's configuration of interrupt mode for full, half-full or every trigger, the interrupt will be generated and the client callback functions will be called from the ISR. The flag oneTimeCallback is passed as an argument for this function. If the value of this flag is TRUE, then the callback will be called only once. If client needs one more callback, he needs to register the callback once more. If this value is false, then whenever interrupt is generated, the callback function will be called until the client call the close function.

Remarks

The registered callback function will be called from ISR. So, it is recommended to keep the callback functions light and not process intensive.

Preconditions

The DRV_CTR_Initialize function must have been called.

DRV_CTR_Open must have been called to obtain a valid opened device handle.

Example

```
#define CLIENT_ID 0x01
DRV_HANDLE      handle;         // Returned from DRV_CTR_Open
void ClientCallack( uintptr_t context, uint32_t * timestampbuffer,
                    uint8_t BufferSize);

DRV_CTR_RegisterCallBack(handle, ClientCallack, FALSE, CLIENT_ID);
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

void DRV_CTR_RegisterCallBack(
const DRV_HANDLE handle,
const DRV_CTR_CALLBACK callback,
const bool oneTimeCallback,

```
const uintptr_t context
);
```

callback - A function pointer for client callback function
oneTimeCallback - If client needs callback to be called only once, then this flag must be true.
context - The value of parameter will be passed back to the client unchanged, when the callback function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

DRV_CTR_TriggerISR Function

Interrupt Service Routine called for the CTR Trigger interrupt.

Implementation: Dynamic

File

drv_ctr.h

C

```
void DRV_CTR_TriggerISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the interrupt is generated for CTR trigger interrupt. The interrupt handling for this interrupt is application specific. So, this function is kept open for the clients to modify.

Remarks

Specific interrupt handling can be taken care of by application developer, as the need for this interrupt is application specific.

Preconditions

None.

Example

This function is not called from clients/system. This function will be called when the interrupt for event is generated.

Parameters

Parameters	Description
object	The driver instance handle returned after the initialization.

Function

void DRV_CTR_TriggerISR(SYS_MODULE_OBJ object);

c) Data Types and Constants

Enumerations

	Name	Description
	DRV_CTR_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_MODE	Defines the driver mode. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_CTR_COUNTER_NUM	Number of counters in CTR module

	DRV_CTR_INDEX_0	CTR driver index definitions
	DRV_CTR_LATCH_FIFO_CNT	FIFO size for each latch in CTR module
	DRV_CTR_LATCH_NUM	Number of latches in CTR module

Structures

	Name	Description
	DRV_CTR_COUNTER	Contains all the data necessary to initialize the CTR counter. **Implementation:** Dynamic
	DRV_CTR_INIT	Contains all the data necessary to initialize the CTR. **Implementation:** Dynamic
	DRV_CTR_LATCH	Contains all the data necessary to initialize the CTR Latches. **Implementation:** Dynamic
	DRV_CTR_TRIGGER	Contains all the data necessary to initialize the CTR Triggers. **Implementation:** Dynamic

Types

	Name	Description
	DRV_CTR_CALLBACK	Callback function definition for CTR event interrupt. **Implementation:** Dynamic

Description

DRV_CTR_CALLBACK Type

Callback function definition for CTR event interrupt.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef void (* DRV_CTR_CALLBACK)(uintptr_t context, uint32_t * timestampbuffer, uint8_t BufferSize);
```

Description

CTR Event interrupt callback function

The clients must define their callback functions in the same prototype as DRV_CTR_CALLBACK. All the registered callbacks will be called from drive ISR for CTR event.

Remarks

This structure is a part of initialization structure, which is used to initialize the CTR module.

DRV_CTR_CLIENT_STATUS Enumeration

Defines the client status.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef enum {
  DRV_CTR_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_CTR_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_CTR_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_CTR_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_CTR_CLIENT_STATUS;
```

Members

Members	Description
DRV_CTR_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_CTR_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_CTR_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_CTR_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

CTR Client Status

Defines the various client status codes.

Remarks

None.

DRV_CTR_COUNTER Structure

Contains all the data necessary to initialize the CTR counter.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef struct {
  uint32_t M;
  uint32_t N;
  uint8_t LSB;
  CTR_MODE_SELECT Mode;
} DRV_CTR_COUNTER;
```

Description

CTR Counter init structure

This structure contains all of the data necessary to initialize the CTR counter increment steps and the resolution.

Remarks

This structure is a part of initialization structure, which is used to initialize the CTR module.

DRV_CTR_INIT Structure

Contains all the data necessary to initialize the CTR.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  CTR_MODULE_ID ctrId;
  INT_SOURCE ctrEventInterruptSource;
  CTR_LATCH_INT_MODE ctrLatchEventMode;
  INT_SOURCE ctrTriggerInterruptSource;
  DRV_CTR_COUNTER ctrCounter[DRV_CTR_COUNTER_NUM];
  DRV_CTR_LATCH ctrLatch[DRV_CTR_LATCH_NUM];
  DRV_CTR_TRIGGER ctrTrigger;
  DRV_MODE drvMode;
} DRV_CTR_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
CTR_MODULE_ID ctrId;	Identifies the CTR peripheral instance
INT_SOURCE ctrEventInterruptSource;	CTR Event Interrupt Source
CTR_LATCH_INT_MODE ctrLatchEventMode;	CTR Event Interrupt Mode
INT_SOURCE ctrTriggerInterruptSource;	CTR Triggetr Interrupt Source
DRV_CTR_COUNTER ctrCounter[DRV_CTR_COUNTER_NUM];	Counter Init Data
DRV_CTR_LATCH ctrLatch[DRV_CTR_LATCH_NUM];	Latch Init Data
DRV_MODE drvMode;	Driver Mode

Description

CTR Driver Initialization Data

This structure contains all of the data necessary to initialize the CTR.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_CTR_Initialize function.

DRV_CTR_LATCH Structure

Contains all the data necessary to initialize the CTR Latches.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef struct {
  CTR_LATCH_TRIGGER_SELECT trigSel;
  CTR_LATCH_CTR_SELECT ctrSel;
  uint8_t divider;
} DRV_CTR_LATCH;
```

Description

CTR Latch init structure

This structure contains all of the data necessary to initialize the CTR Latches for mapping the trigger source and counter for a given latch.

Remarks

This structure is a part of initialization structure, which is used to initialize the CTR module.

DRV_CTR_TRIGGER Structure

Contains all the data necessary to initialize the CTR Triggers.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef struct {
  CTR_LATCH_CTR_SELECT trigSource;
  uint16_t phase;
} DRV_CTR_TRIGGER;
```

Description

CTR Trigger init structure

This structure contains all of the data necessary to initialize the CTR Triggers for generating triggers from CTR.

Remarks

This structure is a part of initialization structure, which is used to initialize the CTR module.

DRV_MODE Enumeration

Defines the driver mode.

Implementation: Dynamic

File

drv_ctr.h

C

```
typedef enum {
  WIFI_MODE = 0,
  USB_MODE,
  GPIO_MODE
} DRV_MODE;
```

Description

CTR Driver mode

Driver can be configured to use for either of Wifi, USB or GPIO.

Remarks

None.

DRV_CTR_COUNTER_NUM Macro

Number of counters in CTR module

File

drv_ctr.h

C

```
#define DRV_CTR_COUNTER_NUM 2
```

Description

Counters present in the CTR module

These constants provide Number of counters in CTR module.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CTR_INDEX_0 Macro

CTR driver index definitions

File

drv_ctr.h

C

```
#define DRV_CTR_INDEX_0 0
```

Description

Driver CTR Module Index reference

These constants provide CTR driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_CTR_Initialize and DRV_CTR_Open routines to identify the driver instance in use.

DRV_CTR_LATCH_FIFO_CNT Macro

FIFO size for each latch in CTR module

File

drv_ctr.h

C

```
#define DRV_CTR_LATCH_FIFO_CNT 4
```

Description

FIFO size for each latch in the CTR module

These constants provide Number of FIFO location available in each latch in CTR module.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CTR_LATCH_NUM Macro

Number of latches in CTR module

File

drv_ctr.h

C

```
#define DRV_CTR_LATCH_NUM 6
```

Description

Latches present in the CTR module

These constants provide Number of latches in CTR module.

Remarks

These constants should be used in place of hard-coded numeric literals.

Files

Files

Name	Description
drv_ctr.h	CTR Driver Interface Definition

Description

This section lists the source and header files used by the CTR Driver Library.

drv_ctr.h

CTR Driver Interface Definition

Enumerations

	Name	Description
	DRV_CTR_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_MODE	Defines the driver mode. **Implementation:** Dynamic

Functions

	Name	Description
	DRV_CTR_Adjust	Sets the adjust value for a given CTR counter. **Implementation:** Dynamic

	DRV_CTR_ClientStatus	Gets current client-specific status of the CTR driver. **Implementation:** Dynamic
	DRV_CTR_Close	Closes an opened-instance of the CTR driver. **Implementation:** Dynamic
	DRV_CTR_Deinitialize	Deinitializes the specified instance of the CTR driver module. **Implementation:** Dynamic
	DRV_CTR_Drift	Sets the drift value for a given CTR counter. **Implementation:** Dynamic
	DRV_CTR_EventISR	Interrupt Service Routine called for the CTR event interrupt. **Implementation:** Dynamic
	DRV_CTR_Initialize	Initializes the CTR Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_CTR_Open	Opens the specified CTR driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_CTR_RegisterCallBack	Registers a callback function for the event interrupt of CTR. **Implementation:** Dynamic
	DRV_CTR_Status	Gets the current status of the CTR Driver module. **Implementation:** Dynamic
	DRV_CTR_TriggerISR	Interrupt Service Routine called for the CTR Trigger interrupt. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_CTR_COUNTER_NUM	Number of counters in CTR module
	DRV_CTR_INDEX_0	CTR driver index definitions
	DRV_CTR_LATCH_FIFO_CNT	FIFO size for each latch in CTR module
	DRV_CTR_LATCH_NUM	Number of latches in CTR module

Structures

	Name	Description
	DRV_CTR_COUNTER	Contains all the data necessary to initialize the CTR counter. **Implementation:** Dynamic
	DRV_CTR_INIT	Contains all the data necessary to initialize the CTR. **Implementation:** Dynamic
	DRV_CTR_LATCH	Contains all the data necessary to initialize the CTR Latches. **Implementation:** Dynamic
	DRV_CTR_TRIGGER	Contains all the data necessary to initialize the CTR Triggers. **Implementation:** Dynamic

Types

	Name	Description
	DRV_CTR_CALLBACK	Callback function definition for CTR event interrupt. **Implementation:** Dynamic

Description

CTR Driver Interface Definition

The CTR device driver provides a simple interface to manage the CTR Module This file defines the interface definition for the CTR Driver.

File Name

drv_CTR.h

Company

Microchip Technology Inc.

Data EEPROM Driver Library

This section describes the Data EEPROM Driver Library.

Introduction

The MPLAB Harmony Data EEPROM Driver provides a high-level interface to manage the Data EEPROM module on the Microchip family of microcontrollers.

Description

The Data EEPROM Driver provides the following features:

- Application-ready routines to perform block operations on the Data EEPROM
- Multi-client operation support
- Data transfer events
- Supports Non-blocking mode of operation only

The Data EEPROM Driver supports multi-client operation, which allows multiple application clients to access the same memory device. Multiple instances of the driver can be used when multiple EEPROM devices are required to be part of the system.

Using the Library

This topic describes the basic architecture of the Data EEPROM Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_eeprom.h`

The interface to the EEPROM Driver Library is defined in the `drv_eeprom.h` header file. Any C language source (`.c`) file that uses the Data EPROM Driver Library should include `drv_eeprom.h`.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

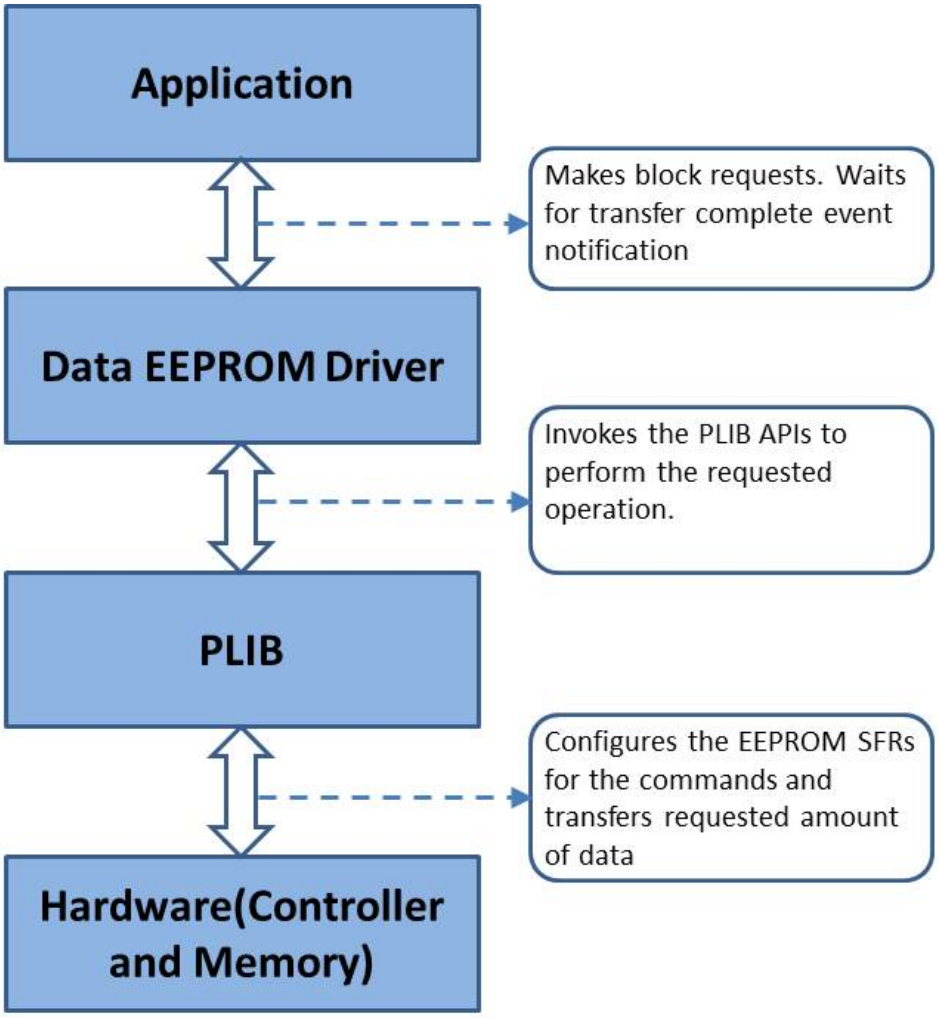
Abstraction Model

This library provides a low-level abstraction of the Data EEPROM Driver Library on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The Data EEPROM driver provides a set of APIs that can be used to perform Erase, Write, and Read operations. The following diagram depicts the communication between different modules. As shown in the diagram, the Data EEPROM Driver sits between the Peripheral Libraries and the application or system layer to facilitate block and file access to the EEPROM.

Data EEPROM Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Data EEPROM Driver.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, tasks, and status functions.
Client Core Functions	Provides open, close, and other setup functions.
Block Operation Functions	Provides read, write, and erase functions to perform data transfer operations on the EEPROM device.
Media Interface Functions	Provides functions to query the EEPROM geometry and media status.

How the Library Works

Provides information on system, client core, block operation, and media interface functions.

Description

System Functions

Data EEPROM Driver Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During

system initialization, each instance of the Data EEPROM Driver would be initialized with the following configuration settings passed dynamically at run time using DRV_EEPROM_INIT, that are supported by the specific EEPROM driver:

- Device requested power state: One of the system module power states. For specific details please refer to "Data Types and Constants" in the Library Interfacesection.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., NVM_ID_0)
- EEPROM Media Geometry

The DRV_EEPROM_Initialize function configures and initializes the EEPROM driver using the configuration information provided. It returns an object handle of the type SYS_MODULE_OBJ. This object handle would be used by other system interfaces such as DRV_EEPROM_Status, DRV_EEPROM_Tasks and DRV_EEPROM_Deinitialize.

Example:

```
/*** Data EEPROM Driver Initialization Data ***/
SYS_FS_MEDIA_REGION_GEOMETRY EEPROMGeometryTable[3] =
{
    {
        .blockSize = 4,
        .numBlocks = (DRV_EEPROM_MEDIA_SIZE * 1024),
    },
    {
       .blockSize = 4,
       .numBlocks = ((DRV_EEPROM_MEDIA_SIZE * 1024)/4)
    },
    {
       .blockSize = 4,
       .numBlocks = ((DRV_EEPROM_MEDIA_SIZE * 1024)/4)
    }
};

const SYS_FS_MEDIA_GEOMETRY EEPROMGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = (SYS_FS_MEDIA_REGION_GEOMETRY *)&EEPROMGeometryTable
};

const DRV_EEPROM_INIT drvEepromInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .eepromId = NVM_ID_0,
    .eepromMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&EEPROMGeometry
};

/* Initialize the Data EEPROM Driver */
sysObj.drvEeprom = DRV_EEPROM_Initialize(DRV_EEPROM_INDEX_0, (SYS_MODULE_INIT *)&drvEepromInit);
```

Data EEPROM Driver Task Routine

The Data EEPROM Driver task routine DRV_EEPROM_Tasks, will be called from the system task routine, SYS_Tasks. The driver task routine is responsible maintaining the driver state machine. The block operation requests from the application or from other modules are added to the driver queue.

Data EEPROM Driver Status

DRV_EEPROM_Status() returns the current status of the Data EEPROM Driver and is called by the MPLAB Harmony System. The application may not find the need to call this function directly.

Example:

```
SYS_MODULE_OBJ object;
// Returned from DRV_EEPROM_Initialize
SYS_STATUS eepromStatus;

eepromStatus = DRV_EEPROM_Status(object);
if (SYS_STATUS_ERROR >= eepromStatus)
{
    // Handle error
}
```

Client Core Functions

Opening the Driver

For the application to start using an instance of the module, it must call the DRV_EEPROM_Open function repeatedly until a valid handle is returned by the driver. The application client uses this driver handle to access the driver functions.

For the various options available for I/O INTENT please refer to Data Types and Constants in the Library Interface section.

Example:

```
eepromHandle = DRV_EEPROM_Open(DRV_EEPROM_INDEX_0, DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == eepromHandle)
{
    /* Call until the function returns a valid handle. */
}
else
{
    /* Do further processing. */
}
```

Closing the Driver

Closes an opened-instance of the Data EEPROM Driver. This invalidates the driver handle. The application must open the driver again to obtain a valid handle.

Example:

```
DRV_HANDLE eepromHandle; // Returned from DRV_EEPROM_Open
DRV_EEPROM_Close(eepromHandle);
```

Client Block Operation Functions

The driver provides client interfaces to perform operations in terms of blocks. A block is a unit that represents the minimum amount of data that can be erased, written, or read. The block sizes may differ for Erase, Write, and Read operations. The DRV_EEPROM_GeometryGet function can be used to read out the geometry of the EEPROM device. The geometry indicates the number of read, write and erase regions, blocks per region and the size of each block.

The DRV_EEPROM_Erase, DRV_EEPROM_Write, and DRV_EEPROM_Read functions are used to erase, write, and read the data to/from EEPROM devices. In addition to these functions, the driver also provides the DRV_EEPROM_BulkErase function that erases the entire EEPROM.

These functions are non-blocking in nature and queue the operation request into the driver queue. All of the requests in the queue are executed by the DRV_EEPROM_Tasks function one-by-one. A command handle associated with the operation request is returned to the application client when the operation request is queued at the driver. This handle allows the application client to track the request as it progresses through the queue. The handle expires when the request processing is complete. The driver provides events (DRV_EEPROM_EVENT) that indicate the completion of the requests.

The following steps can be performed for a simple Block Data Operation:

1. The system should have completed necessary initialization of the Data EEPROM Driver, and the DRV_EEPROM_Tasks function should be running in a polled environment.
2. Open the driver using DRV_EEPROM_Open with the necessary intent.
3. Set an event handler callback using the function DRV_EEPROM_EventHandlerSet.
4. Request for block operations using the functions, DRV_EEPROM_Erase, DRV_EEPROM_Write, DRV_EEPROM_Read and DRV_EEPROM_BulkErase with the appropriate parameters.
5. Wait for event handler callback to occur and check the status of the block operation using the callback function parameter of type DRV_EEPROM_ EVENT.
6. After performing the required block operations, the client can close the driver using the function , DRV_EEPROM_Close .

Example:

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_EEPROM_COMMAND_HANDLE commandHandle;

// drvEEPROMHandle is the handle returned by the DRV_EEPROM_Open // function. Client registers an event
handler with driver. This // is done once.

DRV_EEPROM_EventHandlerSet(drvEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);

DRV_EEPROM_Read(drvEEPROMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation // is done.
```

```
void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event
    // handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:
            // Operation completed successfully.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Media Interface Functions

Reading the Device Geometry

The application can call the DRV_EEPROM_GeometryGet function to obtain the geometry of the EEPROM device. The geometry indicates the number of read, write and erase regions, number of blocks per region and the size of each block.

Example:

```
SYS_FS_MEDIA_GEOMETRY * eepromGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalSize;

eepromGeometry = DRV_EEPROM_GeometryGet(eepromOpenHandle1);

readBlockSize = eepromGeometry->geometryTable->blockSize;
nReadBlocks = eepromGeometry->geometryTable->numBlocks;
nReadRegions = eepromGeometry->numReadRegions;

writeBlockSize = (eepromGeometry->geometryTable +1)->blockSize;
eraseBlockSize = (eepromGeometry->geometryTable +2)->blockSize;

//The below expression provides the EEPROM memory size.
totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Configuring the Library

Macros

	Name	Description
	DRV_EEPROM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_EEPROM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_EEPROM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_EEPROM_MEDIA_SIZE	Specifies the EEPROM Media size.
	DRV_EEPROM_SYS_FS_REGISTER	Register to use with the File system

Description

The configuration of the Data EEPROM Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the Data EEPROM Driver. Based on the selections made, the Data EEPROM Driver may

support the selected features. These configuration settings will apply to all instances of the Data EEPROM Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_EEPROM_BUFFER_OBJECT_NUMBER Macro

Selects the maximum number of buffer objects

File

drv_eeprom_config_template.h

C

```
#define DRV_EEPROM_BUFFER_OBJECT_NUMBER 5
```

Description

EEPROM Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The EEPROM Driver can queue up to DRV_EEPROM_BUFFER_OBJECT_NUMBER of read/write requests before returning a DRV_EEPROM_BUFFER_HANDLE_INVALID due to the queue being full. Buffer objects are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_EEPROM_CLIENTS_NUMBER Macro

Selects the maximum number of clients

File

drv_eeprom_config_template.h

C

```
#define DRV_EEPROM_CLIENTS_NUMBER 1
```

Description

EEPROM maximum number of clients

This definition selects the maximum number of clients that the EEPROM driver can support at run time. This constant defines the total number of EEPROM driver clients that will be available to all instances of the EEPROM driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_EEPROM_INSTANCES_NUMBER Macro

Selects the maximum number of Driver instances that can be supported by the dynamic driver.

File

drv_eeprom_config_template.h

C

```
#define DRV_EEPROM_INSTANCES_NUMBER 1
```

Description

EEPROM Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_EEPROM_MEDIA_SIZE Macro

Specifies the EEPROM Media size.

File

drv_eeprom_config_template.h

C

```
#define DRV_EEPROM_MEDIA_SIZE 32
```

Description

EEPROM Media Size

This definition specifies the EEPROM Media Size to be used. The size is specified in number of Kilo Bytes. The media size MUST never exceed physical available EEPROM Memory size. Application code requirements should be kept in mind while defining this parameter.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_EEPROM_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_eeprom_config_template.h

C

```
#define DRV_EEPROM_SYS_FS_REGISTER
```

Description

EEPROM Driver Register with File System

Specifying this macro enables the EEPROM driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the EEPROM driver is to be used with the File System.

Building the Library

This section lists the files that are available in the Data EEPROM Driver Library.

Description

This section lists the files that are available in the `\src` folder of the Data EEPROM Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/eeprom`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_eeprom.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_eeprom.c`	Basic Data EEPROM Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Data EEPROM Driver Library is not dependent upon any modules.

Library Interface

a) System Functions

	Name	Description
	DRV_EEPROM_Initialize	Initializes the EEPROM instance for the specified driver index.
	DRV_EEPROM_Deinitialize	Deinitializes the specified instance of the EEPROM driver module
	DRV_EEPROM_Status	Gets the current status of the EEPROM driver module.
	DRV_EEPROM_Tasks	Handles the read or write requests queued to the driver.

b) Client Core Functions

	Name	Description
	DRV_EEPROM_Close	Closes an opened-instance of the EEPROM driver
	DRV_EEPROM_Open	Opens the specified EEPROM driver instance and returns a handle to it

c) Block Operation Functions

	Name	Description
	DRV_EEPROM_BulkErase	Performs a bulk erase of the entire Data EEPROM.
	DRV_EEPROM_Erase	Erases blocks of data starting from the specified block address.
	DRV_EEPROM_Read	Reads blocks of data from the specified address in EEPROM memory.
	DRV_EEPROM_Write	Writes blocks of data starting from the specified address in EEPROM memory.

d) Media Interface Functions

	Name	Description
	DRV_EEPROM_AddressGet	Returns the EEPROM media start address
	DRV_EEPROM_CommandStatus	Gets the current status of the command.
	DRV_EEPROM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_EEPROM_GeometryGet	Returns the geometry of the device.
	DRV_EEPROM_IsAttached	Returns the physical attach status of the EEPROM.
	DRV_EEPROM_IsWriteProtected	Returns the write protect status of the EEPROM.

e) Data Types and Constants

	Name	Description
	DRV_EEPROM_COMMAND_HANDLE_INVALID	This value defines the EEPROM Driver's Invalid Command Handle.
	DRV_EEPROM_INDEX_0	EEPROM driver index definition
	DRV_EEPROM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_EEPROM_COMMAND_STATUS	Specifies the status of the command for read or write requests.
	DRV_EEPROM_EVENT	Identifies the possible events that can result from a request.
	DRV_EEPROM_EVENT_HANDLER	Pointer to a EEPROM Driver Event handler function
	DRV_EEPROM_INIT	Defines the data required to initialize the EEPROM driver

Description

This section describes the Application Programming Interface (API) functions of the Data EEPROM Driver Library.

a) System Functions

DRV_EEPROM_Initialize Function

Initializes the EEPROM instance for the specified driver index.

File

drv_eeprom.h

C

```
SYS_MODULE_OBJ DRV_EEPROM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the EEPROM driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other EEPROM routine is called.

This routine should only be called once during system initialization unless DRV_EEPROM_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. The system must use DRV_EEPROM_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the EEPROM Driver.

SYS_MODULE_OBJ  objectHandle;

SYS_FS_MEDIA_REGION_GEOMETRY EEPROMGeometryTable[3] =
{
    {
        .blockSize = 4,
        .numBlocks = (DRV_EEPROM_MEDIA_SIZE * 1024),
    },
    {
        .blockSize = 4,
        .numBlocks = ((DRV_EEPROM_MEDIA_SIZE * 1024)/4)
    },
    {
        .blockSize = 4,
        .numBlocks = ((DRV_EEPROM_MEDIA_SIZE * 1024)/4)
    }
};

const SYS_FS_MEDIA_GEOMETRY EEPROMGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = (SYS_FS_MEDIA_REGION_GEOMETRY *)&EEPROMGeometryTable
};

// EEPROM Driver Initialization Data
const DRV_EEPROM_INIT drvEepromInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .eepromId = NVM_ID_0,
    .eepromMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&EEPROMGeometry
};
```

```
objectHandle = DRV_EEPROM_Initialize(DRV_EEPROM_INDEX_0, (SYS_MODULE_INIT*)&drvEepromInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized.
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_EEPROM_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_EEPROM_Deinitialize Function

Deinitializes the specified instance of the EEPROM driver module

File

drv_eeprom.h

C

```
void DRV_EEPROM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the EEPROM driver module, disabling its operation. Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_EEPROM_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_EEPROM_Initialize routine

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_EEPROM_Initialize
SYS_STATUS          status;

DRV_EEPROM_Deinitialize(object);

status = DRV_EEPROM_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know when the driver is
    // deinitialized.
}
```

Function

void DRV_EEPROM_Deinitialize

(

SYS_MODULE_OBJ object

);

DRV_EEPROM_Status Function

Gets the current status of the EEPROM driver module.

File

drv_eeprom.h

C

```
SYS_STATUS DRV_EEPROM_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations.

SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the EEPROM driver module.

Remarks

None.

Preconditions

Function DRV_EEPROM_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_EEPROM_Initialize
SYS_STATUS          EEPROMStatus;

EEPROMStatus = DRV_EEPROM_Status(object);
if (EEPROMStatus == SYS_STATUS_READY)
{
    // Driver is ready to perform operations.
}
else
{
    // Driver is not ready.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_EEPROM_Initialize routine

Function

SYS_STATUS DRV_EEPROM_Status

(

SYS_MODULE_OBJ object

);

DRV_EEPROM_Tasks Function

Handles the read or write requests queued to the driver.

File

drv_eeprom.h

C

```
void DRV_EEPROM_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to handle the read or write requests queued to the driver.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_EEPROM_Initialize routine must have been called for the specified EEPROM driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_EEPROM_Initialize

while (true)
{
    DRV_EEPROM_Tasks (object);
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_EEPROM_Initialize)

Function

void DRV_EEPROM_Tasks
(
SYS_MODULE_OBJ object
);

b) Client Core Functions

DRV_EEPROM_Close Function

Closes an opened-instance of the EEPROM driver

File

drv_eeprom.h

C

```
void DRV_EEPROM_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the EEPROM driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_EEPROM_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_EEPROM_Initialize routine must have been called for the specified EEPROM driver instance.

DRV_EEPROM_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_EEPROM_Open

DRV_EEPROM_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_EEPROM_Close
(
const DRV_HANDLE handle
);

DRV_EEPROM_Open Function

Opens the specified EEPROM driver instance and returns a handle to it

File

drv_eeprom.h

C

```
DRV_HANDLE DRV_EEPROM_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_EEPROM_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver hardware instance being opened is invalid

Description

This routine opens the specified EEPROM driver instance and provides a handle. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_EEPROM_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has already been opened, it cannot be opened exclusively.

Preconditions

DRV_EEPROM_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_EEPROM_Open(DRV_EEPROM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_EEPROM_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

c) Block Operation Functions

DRV_EEPROM_BulkErase Function

Performs a bulk erase of the entire Data EEPROM.

File

drv_eeprom.h

C

```
void DRV_EEPROM_BulkErase(const DRV_HANDLE handle, DRV_EEPROM_COMMAND_HANDLE * commandHandle);
```

Returns

If the request was queued successfully then a valid command handle is returned in the commandHandle argument. Otherwise DRV_EEPROM_COMMAND_HANDLE_INVALID is returned if the request was not successful.

Description

This function schedules a non-blocking bulk erase operation of the entire Data EEPROM. The function returns with a valid handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_EEPROM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the client opened the driver for read only
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_EEPROM_EVENT_COMMAND_COMPLETE event if the command was processed successfully or DRV_EEPROM_EVENT_COMMAND_ERROR event if the command was not processed successfully.

Remarks

None

Refer to drv_eeprom.h for usage information.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
DRV_EEPROM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myEEPROMHandle is the handle returned by the DRV_EEPROM_Open function.
// Client registers an event handler with driver

DRV_EEPROM_EventHandlerSet(myEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);
DRV_EEPROM_BulkErase(myEEPROMHandle, &commandHandle);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
```

```
    // context points to myAppObj.
    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // Bulk Erase operation is complete.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:

            // Bulk Erase operation failed.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle

Function

void DRV_EEPROM_BulkErase

(

const DRV_HANDLE handle,

DRV_EEPROM_COMMAND_HANDLE * commandHandle

);

DRV_EEPROM_Erase Function

Erases blocks of data starting from the specified block address.

File

drv_eeprom.h

C

```
void DRV_EEPROM_Erase(const DRV_HANDLE handle, DRV_EEPROM_COMMAND_HANDLE * commandHandle, uint32_t
blockStart, uint32_t nBlock);
```

Returns

If the request was queued successfully then a valid command handle is returned in the commandHandle argument. Otherwise DRV_EEPROM_COMMAND_HANDLE_INVALID is returned if the request was not successful.

Description

This function schedules a non-blocking erase operation for erasing blocks of memory. The function returns with a valid handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_EEPROM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the client opened the driver for read only
- if the number of blocks to be erased is either zero or more than the number of blocks actually available
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_EEPROM_EVENT_COMMAND_COMPLETE event if the command was processed successfully or DRV_EEPROM_EVENT_COMMAND_ERROR event if the command was not processed successfully.

Remarks

None

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
uint32_t blockStart = 0;
uint32_t nBlock = 2;
DRV_EEPROM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myEEPROMHandle is the handle returned by the DRV_EEPROM_Open function.
// Client registers an event handler with driver

DRV_EEPROM_EventHandlerSet(myEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);
DRV_EEPROM_Erase(myEEPROMHandle, &commandHandle, blockStart, nBlock);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    // context points to myAppObj.
    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // Erase operation is complete.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:

            // Erase operation failed.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	block start addess for the erase operation.
nBlock	Total number of blocks to be erased.

Function

void DRV_EEPROM_Erase

(

const DRV_HANDLE handle,

DRV_EEPROM_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,

uint32_t nBlock

);

DRV_EEPROM_Read Function

Reads blocks of data from the specified address in EEPROM memory.

File

drv_eeprom.h

C

```
void DRV_EEPROM_Read(const DRV_HANDLE handle, DRV_EEPROM_COMMAND_HANDLE * commandHandle, void * buffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

If the request was queued successfully then a valid command handle is returned in the commandHandle argument. Otherwise DRV_EEPROM_COMMAND_HANDLE_INVALID is returned if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from the EEPROM memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the driver instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_EEPROM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the buffer pointer is NULL
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid
- if the number of blocks to be read is zero or more than the actual number of blocks available
- if the client opened the driver in write only mode

If the requesting client registered an event callback with the driver, the driver will issue a DRV_EEPROM_EVENT_COMMAND_COMPLETE event if the command was processed successfully or DRV_EEPROM_EVENT_COMMAND_ERROR event if the command was not processed successfully.

Remarks

None.

Preconditions

The DRV_EEPROM_Initialize routine must have been called for the specified EEPROM driver instance.

DRV_EEPROM_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];
// address should be block aligned.
uint32_t blockStart = EEPROM_BASE_ADDRESS_TO_READ_FROM;
uint32_t nBlock = 2;
DRV_EEPROM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myEEPROMHandle is the handle returned by the DRV_EEPROM_Open function.
DRV_EEPROM_EventHandlerSet(myEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);
DRV_EEPROM_Read(myEEPROMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read queued successfully.
}

// Event is received when the buffer is processed.

void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
```

```
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    // context points to myAppObj.

    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
buffer	Buffer into which the data read from the EEPROM memory will be placed
blockStart	Start block address in EEPROM memory from where the read should begin.
nBlock	Total number of blocks to be read.

Function

void DRV_EEPROM_Read
(
const DRV_HANDLE handle,
DRV_EEPROM_COMMAND_HANDLE * commandHandle,
void * buffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_EEPROM_Write Function

Writes blocks of data starting from the specified address in EEPROM memory.

File

drv_eeprom.h

C

```
void DRV_EEPROM_Write(const DRV_HANDLE handle, DRV_EEPROM_COMMAND_HANDLE * commandHandle, void * buffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

If the request was queued successfully then a valid command handle is returned in the commandHandle argument. Otherwise DRV_EEPROM_COMMAND_HANDLE_INVALID is returned if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into memory. The function returns with a valid handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_EEPROM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the buffer pointer is NULL

- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_EEPROM_EVENT_COMMAND_COMPLETE event if the command was processed successfully or DRV_EEPROM_EVENT_COMMAND_ERROR event if the command was not processed successfully.

Remarks

None

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

uint32_t blockStart = EEPROM_BASE_ADDRESS_TO_WRITE_TO;
uint32_t nBlock = 2;
DRV_EEPROM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myEEPROMHandle is the handle returned by the DRV_EEPROM_Open function.
// Client registers an event handler with driver

DRV_EEPROM_EventHandlerSet(myEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);
DRV_EEPROM_Write(myEEPROMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    // context points to myAppObj.
    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle

buffer	The buffer containing data to be programmed into EEPROM memory
blockStart	Start block address of EEPROM memory where the write should begin.
nBlock	Total number of blocks to be written.

Function

```
void DRV_EEPROM_Write
(
const       DRV_HANDLE handle,
    DRV_EEPROM_COMMAND_HANDLE * commandHandle,
void * buffer,
uint32_t blockStart,
uint32_t nBlock
);
```

d) Media Interface Functions

DRV_EEPROM_AddressGet Function

Returns the EEPROM media start address

File

drv_eeprom.h

C

```
uintptr_t DRV_EEPROM_AddressGet(const DRV_HANDLE handle);
```

Returns

Start address of the EEPROM Media if the handle is valid otherwise NULL.

Description

This function returns the EEPROM Media start address.

Remarks

None.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

The DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
uintptr_t startAddress;
startAddress = DRV_EEPROM_AddressGet(drvEEPROMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

```
uintptr_t DRV_EEPROM_AddressGet
(
const       DRV_HANDLE handle
);
```

DRV_EEPROM_CommandStatus Function

Gets the current status of the command.

File

drv_eeprom.h

C

```
DRV_EEPROM_COMMAND_STATUS DRV_EEPROM_CommandStatus(const DRV_HANDLE handle, const DRV_EEPROM_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_EEPROM_COMMAND_STATUS value describing the current status of the command.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to be polled on. The function may return DRV_EEPROM_COMMAND_HANDLE_INVALID in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another read, write or erase request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive read, write or erase operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called.

The DRV_EEPROM_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                handle;         // Returned from DRV_EEPROM_Open
DRV_EEPROM_COMMAND_HANDLE commandHandle;
DRV_EEPROM_COMMAND_STATUS status;

status = DRV_EEPROM_CommandStatus(handle, commandHandle);
if(status == DRV_EEPROM_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
commandHandle	A valid command handle returned from read, write or erase request.

Function

DRV_EEPROM_COMMAND_STATUS DRV_EEPROM_CommandStatus

(

const DRV_HANDLE handle,

const DRV_EEPROM_COMMAND_HANDLE commandHandle

);

DRV_EEPROM_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_eeprom.h

C

```
void DRV_EEPROM_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls a read, write or a erase function, it is provided with a handle identifying the command that was added to the driver's command queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read, write or erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

The DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_EEPROM_COMMAND_HANDLE commandHandle;

// drvEEPROMHandle is the handle returned by the DRV_EEPROM_Open function.
// Client registers an event handler with driver. This is done once.

DRV_EEPROM_EventHandlerSet(drvEEPROMHandle, APP_EEPROMEventHandler, (uintptr_t)&myAppObj);

DRV_EEPROM_Read(drvEEPROMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_EEPROM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_EEPROMEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_EEPROM_EventHandlerSet
(
const DRV_HANDLE handle,
const void * eventHandler,
const uintptr_t context
);

DRV_EEPROM_GeometryGet Function

Returns the geometry of the device.

File

drv_eeprom.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_EEPROM_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the EEPROM memory:

- Media Property
- Number of Read/Write/Erase regions
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

The DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * eepromGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalSize;

eepromGeometry = DRV_EEPROM_GeometryGet(eepromOpenHandle1);

readBlockSize  = eepromGeometry->geometryTable->blockSize;
nReadBlocks = eepromGeometry->geometryTable->numBlocks;
nReadRegions = eepromGeometry->numReadRegions;

writeBlockSize  = (eepromGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (eepromGeometry->geometryTable +2)->blockSize;

totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_EEPROM_GeometryGet
(
const DRV_HANDLE handle
);

DRV_EEPROM_IsAttached Function

Returns the physical attach status of the EEPROM.

File

drv_eeprom.h

C

```
bool DRV_EEPROM_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns true always

Description

This function returns the physical attach status of the EEPROM.

Remarks

None.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

The DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The EEPROM media is always attached and so the below always returns
// true.

bool isEEPROMAttached;
isEEPROMAttached = DRV_EEPROM_isAttached(drvEEPROMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_EEPROM_IsAttached
(
const DRV_HANDLE handle
);

DRV_EEPROM_IsWriteProtected Function

Returns the write protect status of the EEPROM.

File

drv_eeprom.h

C

```
bool DRV_EEPROM_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Always returns false.

Description

This function returns the physical attach status of the EEPROM. This function always returns false.

Remarks

None.

Preconditions

The DRV_EEPROM_Initialize() routine must have been called for the specified EEPROM driver instance.

The DRV_EEPROM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The EEPROM media is treated as always writeable.
bool isWriteProtected;
isWriteProtected = DRV_EEPROM_IsWriteProtected(drvEEPROMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_EEPROM_IsWriteProtected

(

const DRV_HANDLE handle

);

e) Data Types and Constants

DRV_EEPROM_COMMAND_HANDLE_INVALID Macro

This value defines the EEPROM Driver's Invalid Command Handle.

File

drv_eeprom.h

C

```
#define DRV_EEPROM_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

EEPROM Driver Invalid Command Handle.

This value defines the EEPROM Driver Invalid Command Handle. This value is returned by read or write routines when the command request was not accepted.

Remarks

None.

DRV_EEPROM_INDEX_0 Macro

EEPROM driver index definition

File

drv_eeprom.h

C

```
#define DRV_EEPROM_INDEX_0 0
```

Description

Driver EEPROM Module Index reference

This constant provides EEPROM driver index definition.

Remarks

This constant should be used in place of hard-coded numeric literals. This value should be passed into the DRV_EEPROM_Initialize and DRV_EEPROM_Open routines to identify the driver instance in use.

DRV_EEPROM_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_eeprom.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_EEPROM_COMMAND_HANDLE;
```

Description

EEPROM Driver command handle.

A command handle is returned by a call to the read or write functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_EEPROM_COMMAND_STATUS Enumeration

Specifies the status of the command for read or write requests.

File

drv_eeprom.h

C

```
typedef enum {
  DRV_EEPROM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_EEPROM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_EEPROM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_EEPROM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_EEPROM_COMMAND_STATUS;
```

Members

Members	Description
DRV_EEPROM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_EEPROM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_EEPROM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_EEPROM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

EEPROM Driver Command Status

EEPROM Driver command Status

This type specifies the status of the command for the read or write requests.

Remarks

None.

DRV_EEPROM_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_eeprom.h

C

```
typedef enum {
  DRV_EEPROM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_EEPROM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_EEPROM_EVENT;
```

Members

Members	Description
DRV_EEPROM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_EEPROM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

EEPROM Driver Events

This enumeration identifies the possible events that can result from a read or write request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_EEPROM_EventHandlerSet function when a request is completed.

DRV_EEPROM_EVENT_HANDLER Type

Pointer to a EEPROM Driver Event handler function

File

drv_eeprom.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_EEPROM_EVENT_HANDLER;
```

Returns

None.

Description

EEPROM Driver Event Handler Function Pointer

This data type defines the required function signature for the EEPROM event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event callbacks from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_EEPROM_EVENT_COMMAND_COMPLETE, it means that the scheduled operation was completed successfully.

If the event is DRV_EEPROM_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_EEPROM_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that scheduled the request.

The event handler function executes in the driver's context. It is recommended of the application to not perform process intensive or blocking operations within this function.

Example

```
void APP_MyEepromEventHandler
(
    DRV_EEPROM_EVENT event,
    DRV_EEPROM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_EEPROM_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_EEPROM_EVENT_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read or Write requests
context	Value identifying the context of the application that registered the event handling function

DRV_EEPROM_INIT Structure

Defines the data required to initialize the EEPROM driver

File

drv_eeprom.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  NVM_MODULE_ID eepromId;
  const SYS_FS_MEDIA_GEOMETRY * eepromMediaGeometry;
} DRV_EEPROM_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
NVM_MODULE_ID eepromId;	Identifies hardware module (PLIB-level) ID
const SYS_FS_MEDIA_GEOMETRY * eepromMediaGeometry;	EEPROM Media geometry object.

Description

EEPROM Driver Initialization Data

This data type defines the data required to initialize the EEPROM driver.

Remarks

None.

Files

Files

Name	Description
drv_eeprom.h	EEPROM Driver Interface Definition
drv_eeprom_config_template.h	EEPROM driver configuration definitions.

Description

drv_eeprom.h

EEPROM Driver Interface Definition

Enumerations

	Name	Description
	DRV_EEPROM_COMMAND_STATUS	Specifies the status of the command for read or write requests.
	DRV_EEPROM_EVENT	Identifies the possible events that can result from a request.

Functions

	Name	Description
	DRV_EEPROM_AddressGet	Returns the EEPROM media start address
	DRV_EEPROM_BulkErase	Performs a bulk erase of the entire Data EEPROM.
	DRV_EEPROM_Close	Closes an opened-instance of the EEPROM driver
	DRV_EEPROM_CommandStatus	Gets the current status of the command.
	DRV_EEPROM_Deinitialize	Deinitializes the specified instance of the EEPROM driver module
	DRV_EEPROM_Erase	Erases blocks of data starting from the specified block address.
	DRV_EEPROM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_EEPROM_GeometryGet	Returns the geometry of the device.
	DRV_EEPROM_Initialize	Initializes the EEPROM instance for the specified driver index.
	DRV_EEPROM_IsAttached	Returns the physical attach status of the EEPROM.
	DRV_EEPROM_IsWriteProtected	Returns the write protect status of the EEPROM.
	DRV_EEPROM_Open	Opens the specified EEPROM driver instance and returns a handle to it
	DRV_EEPROM_Read	Reads blocks of data from the specified address in EEPROM memory.
	DRV_EEPROM_Status	Gets the current status of the EEPROM driver module.
	DRV_EEPROM_Tasks	Handles the read or write requests queued to the driver.
	DRV_EEPROM_Write	Writes blocks of data starting from the specified address in EEPROM memory.

Macros

	Name	Description
	DRV_EEPROM_COMMAND_HANDLE_INVALID	This value defines the EEPROM Driver's Invalid Command Handle.
	DRV_EEPROM_INDEX_0	EEPROM driver index definition

Structures

	Name	Description
	DRV_EEPROM_INIT	Defines the data required to initialize the EEPROM driver

Types

	Name	Description
	DRV_EEPROM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_EEPROM_EVENT_HANDLER	Pointer to a EEPROM Driver Event handler function

Description

EEPROM Driver Interface Definition

The EEPROM driver provides a simple interface to manage the EEPROM Memory on Microchip microcontrollers. This file defines the interface

definition for the EEPROM driver.

File Name

drv_eeprom.h

Company

Microchip Technology Inc.

drv_eeprom_config_template.h

EEPROM driver configuration definitions.

Macros

	Name	Description
	DRV_EEPROM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_EEPROM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_EEPROM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_EEPROM_MEDIA_SIZE	Specifies the EEPROM Media size.
	DRV_EEPROM_SYS_FS_REGISTER	Register to use with the File system

Description

EEPROM Driver Configuration Template Header file.

This template file describes all the mandatory and optional configuration macros that are needed for building the EEPROM driver. Do not include this file in source code.

File Name

drv_eeprom_config_template.h

Company

Microchip Technology Inc.

ENC28J60 Driver Library Help

This section provides information on the ENC28J60 Driver Library.

Introduction

This library provides a driver-level abstraction of the ENC28J60 integrated Ethernet MAC and 10Base-T PHY that can be connected to the PIC32. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module for details.

The "Host-To-Network"_layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section.

Description

The ENC28J60 External MAC and PHY is an external module to the PIC32 that is connected through a Serial Peripheral Interface (SPI). This driver interfaces with the SPI driver to communicate with the external device to implement a complete Ethernet node in a system.

The following are some of the key features of this module:

- Supports 10 Mbps physical-to-physical layer Ethernet data transfer
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Hardware flow control for both Full and Half-Duplex mode
- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Supports Packet Payload Checksum calculation
- CRC Check
- Supports SPI interface

Using the Library

This topic describes the basic architecture and functionality of the software driver for the ENC28J60 stand-alone Ethernet Controller with SPI, and is meant for advanced users or TCP/IP stack driver developers.

Description

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system-wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

Interface Header File: `drv_enc28j60.h`

The interface to the ENC28J60 Driver Library is defined in the `drv_enc28j60.h` header file. Any C language source (`.c`) file that uses the ENC28J60 Driver Library should include `drv_enc28j60.h`.

Library File: The ENC28J60 Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

The ENC28J60 Driver Library provides the low-level abstraction of the communications protocol to communicate to the ENC28J60 external MAC though the SPI peripheral on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the ENC28J60 Driver Library interface.

Description

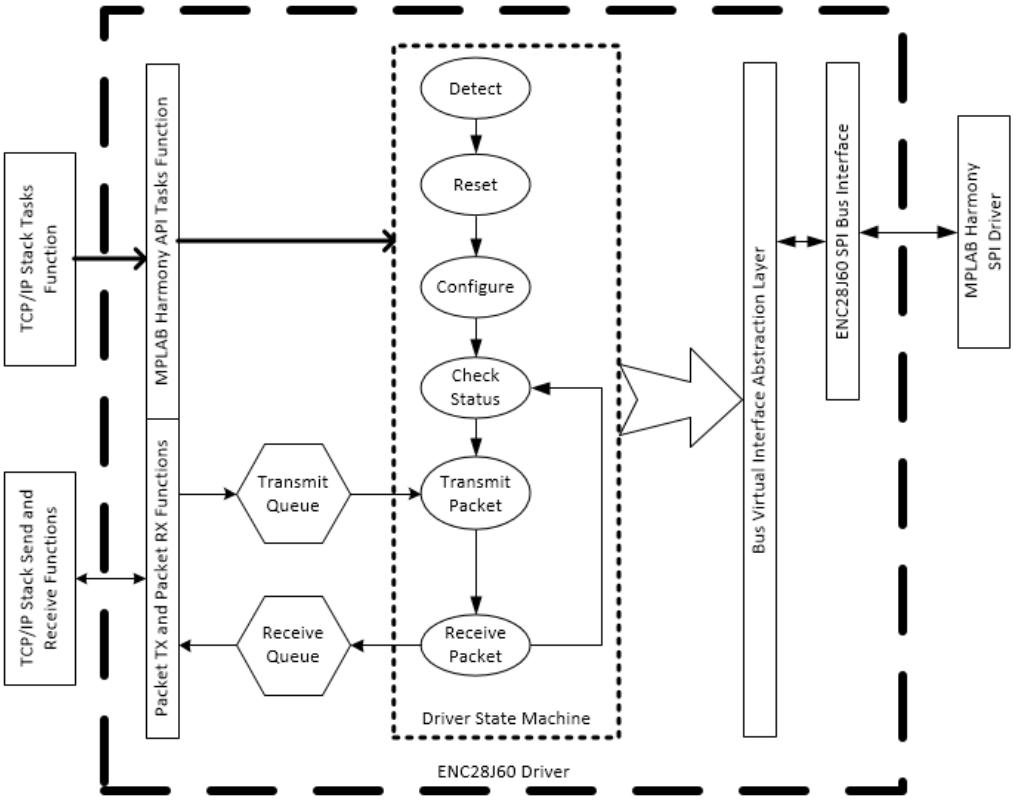
The ENC28J60 Driver library has several different layers to it, as illustrated in the following figure. The interface layer has two main sections that are used the most often: The Tasks function, and the TCP/IP Send and Receive functions.

The Tasks function manages the internal state machine which detects, resets, and then configures the ENC28J60 External MAC. It also handles the monitoring of the hardware status, sending and receiving packets.

The TCP/IP Send and Receive functions interact with the RAM-based queue of packets that are queued to send and packets that have been queued waiting for pick-up by the stack.

The main state machine does not interface directly to the SPI bus, but instead, interfaces to a virtual bus abstraction layer that allows for the replacement of the specific underlying bus implementation.

Abstraction Model



Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the ENC28J60 Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Status Functions	Provides status functions.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support the TCP/IP virtual MAC interface.

Configuring the SPI Driver

This section describes the configuration settings for the ENC28J60 Driver Library.

Description

Configuration

The ENC hardware requires a specific configuration of the SPI driver to work correctly. Inside the MHC SPI Driver configuration be sure to select:

- Run the SPI at frequencies of at least 8 MHz
- Clock mode of DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
- Input phase of SPI_INPUT_SAMPLING_PHASE_AT_END

Recommended Settings

- Interrupt Driver mode
- Enhanced Buffer mode
- DMA mode enabled:
- DMA block transfer size of at least 1600 bytes
- Size of DMA buffer for dummy data of at least 1600 bytes
- Ensure when setting up DMA in interrupt mode that the DMA interrupts are a higher priority than the SPI Driver interrupt

Example:

```
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER           1
#define DRV_SPI_CLIENTS_NUMBER             1
#define DRV_SPI_ELEMENTS_PER_QUEUE         30

/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE             2048
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE      2048

/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0              SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0           DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0            DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0      false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0          SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0           CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0           13333333
#define DRV_SPI_BUFFER_TYPE_IDX0         DRV_SPI_BUFFER_TYPE_ENHANCED
#define DRV_SPI_CLOCK_MODE_IDX0          DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0         SPI_INPUT_SAMPLING_PHASE_AT_END
```

```
#define DRV_SPI_TX_INT_SOURCE_IDX0      INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0      INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0   INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0         INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0       INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0   INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0         30
#define DRV_SPI_RESERVED_JOB_IDX0       1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0     DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0   16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0     DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0   16 Driver Library
```

Configuring the Library

Macros

	Name	Description
	DRV_ENC28J60_CLIENT_INSTANCES	Selects the maximum number of clients.
	DRV_ENC28J60_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

The configuration of the ENC28J60 Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the ENC28J60 Driver Library. Based on the selections made, the ENC28J60 Driver Library may support the selected features. These configuration settings will apply to all instances of the ENC28J60 Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ENC28J60_CLIENT_INSTANCES Macro

Selects the maximum number of clients.

File

drv_enc28j60_config_template.h

C

```
#define DRV_ENC28J60_CLIENT_INSTANCES 1
```

Description

enc28j60 maximum number of clients

This definition selects the maximum number of clients that the enc28j60 driver can support at run-time.

Remarks

Mandatory definition.

DRV_ENC28J60_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_enc28j60_config_template.h

C

```
#define DRV_ENC28J60_INSTANCES_NUMBER 1
```

Description

enc28j60 hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

Mandatory definition.

Building the Library

This section lists the files that are available in the ENC28J60 Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/enc28j60`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source Folder Name	Description
/drv_enc28j60.h	This file provides the interface definitions of the ENC28J60 Driver.

Required File(s)

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

All of the required files listed in the following table are automatically loaded into the MPLAB X IDE project by the MHC.

Source Folder Name	Description
/src/dynamic/drv_drv_enc28j60_api.c	This file contains the API function implementations.
/src/dynamic/drv_enc28j60_main_state.c	This file contains the main state machine functions.
/src/dynamic/drv_enc28j60_utils.c	This file contains functions that are used throughout the driver.
/src/dynamic/bus/spi/drv_enc28j60_spi_bus.c	This file contains the functions to interface with the SPI bus.
/src/dynamic/closed_state/drv_enc28j60_closed_state.c	This file contains the functions for handling the driver closed state.
/src/dynamic/initialization_state/drv_enc28j60_configure_state.c	This file contains the functions for configuring the ENC hardware.
/src/dynamic/initialization_state/drv_enc28j60_detect_state.c	This file contains the functions for detecting the ENC hardware.
/src/dynamic/initialization_state/drv_enc28j60_initialization_state.c	This file contains the functions for the initialization state machine.
/src/dynamic/initialization_state/drv_enc28j60_reset_state.c	This file contains the functions for resetting the ENC hardware.
/src/dynamic/packet/drv_enc28j60_rx_packet.c	This file contains the functions for receiving a packet from the ENC hardware.
/src/dynamic/packet/drv_enc28j60_tx_packet.c	This file contains the functions for sending a packet to the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_change_duplex_state.c	This file contains the functions for configuring the duplex mode of the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_check_int_state.c	This file contains the functions for checking and processing the ENC hardware interrupts.
/src/dynamic/running_state/drv_enc28j60_check_status_state.c	This file contains the functions for checking the status of the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_check_tx_status_state.c	This file contains the functions for checking the status of a transmitted packet.
/src/dynamic/running_state/drv_enc28j60_running_state.c	This file contains the functions for managing the running state machine.
/src/dynamic/running_state/drv_enc28j60_reset_rx_state.c	This file contains the functions for managing the RX state machine reset requirement during run-time.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source Folder Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The ENC28J60 Driver Library depends on the following modules:

- SPI Driver Library
- TCP/IP Stack Library
- TCP/IP Stack MAC Driver Module

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_ENC28J60_Deinitialize	Deinitializes the ENC28J60 Driver Instance. **Implementation:** Dynamic
	DRV_ENC28J60_Initialize	Initializes the ENC28J60 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENC28J60_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENC28J60_Reinitialize	Reinitializes the instance of the ENC28J60 driver. **Implementation:** Dynamic
	DRV_ENC28J60_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENC28J60_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENC28J60_Tasks	Main task function for the driver. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ENC28J60_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENC28J60_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENC28J60_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENC28J60_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENC28J60_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENC28J60_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENC28J60_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENC28J60_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Status	Gets the current status of the driver. **Implementation:** Dynamic

c) Receive Functions

	Name	Description
	DRV_ENC28J60_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic

	DRV_ENC28J60_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic

d) Transmit Functions

	Name	Description
	DRV_ENC28J60_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic

e) Event Functions

	Name	Description
	DRV_ENC28J60_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENC28J60_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENC28J60_EventPendingGet	Gets the current events. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	_DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.
	DRV_ENC28J60_MACObject	ENC28J60 External MAC Virtualization Table

Description

This section describes the Application Programming Interface (API) functions of the ENC28J60 Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_ENC28J60_Deinitialize Function

Deinitializes the ENC28J60 Driver Instance.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENC28J60 Deinitialization

This function deallocates any resources allocated by the initialization function.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
Object	the valid object returned from DRV_ENC28J60_Initialize

DRV_ENC28J60_Initialize Function

Initializes the ENC28J60 Driver Instance, with the configuration data.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_MODULE_OBJ DRV_ENC28J60_Initialize(SYS_MODULE_INDEX index, SYS_MODULE_INIT * init);
```

Returns

- Valid handle to the driver instance - If successful
- SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENC28J60 Initialization

This function initializes the ENC28J60 Driver with configuration data passed into it by either the system_init function or by the DRV_ENC28J60_StackInitialize function. Calling this function alone is not enough to initialize the driver, DRV_ENC28J60_SetMacCtrlInfo must be called with valid data before the driver is ready to be opened.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a DRV_ENC28J60_CONFIG structure.

DRV_ENC28J60_Process Function

Additional processing that happens outside the tasks function.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Process

This function does additional processing that is not done inside the tasks function.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Reinitialize Function

Reinitializes the instance of the ENC28J60 driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

ENC28J60 Reinitialization

This function will deinitialize and initialize the driver instance. As with DRV_ENC28J60_Initialize DRV_ENC28J60_SetMacCtrlInfo must be called for the driver to be useful.

Remarks

This function is not planned to be implemented for the first release.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

DRV_ENC28J60_SetMacCtrlInfo Function

This function sets the MAC control information for the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_SetMacCtrlInfo(SYS_MODULE_OBJ object, TCPIP_MAC_MODULE_CTRL * init);
```

Returns

None.

Description

ENC28J60 Set MAC Control Information

This function is used to pass in the TCPIP_MAC_CONTROL_INIT information that is used for allocation and deallocation of memory, event signaling, etc. This function is needed to be called so that the driver can enter initialization state when the tasks function is called.

Preconditions

The driver had to be successfully initialized with ENC28J60_Initialize.

DRV_ENC28J60_StackInitialize Function

This function initializes the driver with a TCPIP_MAC_INIT object.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_MODULE_OBJ DRV_ENC28J60_StackInitialize(SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

Returns a valid handle to the driver instance - If successful SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENC28J60 Stack Initialization

This function is used by the TCP/IP stack to fully initialize the driver with both the ENC28J60 specific configuration and the MAC control information. With this function there is no need to call DRV_ENC28J60_SetMacCtrlInfo.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a TCPIP_MAC_INIT structure.

DRV_ENC28J60_Tasks Function

Main task function for the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENC28J60 Tasks

This function will execute the main state machine for the ENC28J60 driver.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENC28J60_Initialize

b) Client Level Functions

DRV_ENC28J60_Close Function

Closes a client handle to the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Close(DRV_HANDLE handle);
```

Returns

None.

Description

ENC28J60 Close

This function closes a handle to the driver. If it is the last client open, the driver will send an RX Disable command to the ENC hardware and move

to the closed state.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
handle	The successfully opened handle

DRV_ENC28J60_ConfigGet Function

Gets the current configuration.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
size_t DRV_ENC28J60_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

Number of bytes copied to the buffer

Description

ENC28J60 Get Configuration

Gets the current configuration.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
configBuff	location to copy the configuration too
buffSize	buffer size
pConfigSize	configuration size needed

DRV_ENC28J60_LinkCheck Function

This function returns the status of the link.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is active
- false - all other times

Description

ENC28J60 Link Check

This function checks the status of the link and returns it to the caller.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Open Function

This function is called by the client to open a handle to a driver instance.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
DRV_HANDLE DRV_ENC28J60_Open(SYS_MODULE_INDEX index, DRV_IO_INTENT intent);
```

Returns

Returns a valid handle - If successful INVALID_HANDLE - If unsuccessful

Description

ENC28J60 Open

The client will call this function to open a handle to the driver. When the first instance is opened than the driver will send the RX enabled command to the ENC hardware.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.
intent	The intent to use when opening the driver. Only exclusive is supported

DRV_ENC28J60_ParametersGet Function

Get the parameters of the device.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OK - if the hMac is valid

Description

ENC28J60 Get Parameters

Get the parameters of the device, which includes that it is an Ethernet device and what it's MAC address is. Users of the ENC28J60 must generate a unique MAC address for each controller used.

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

pMacParams	pointer to put the parameters

DRV_ENC28J60_PowerMode Function

This function sets the power mode of the device.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_PowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- false - This functionality is not supported in this version of the driver

Description

ENC28J60 Power Mode

This function sets the power mode of the ENC28J60.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pwrMode	the power mode to set

DRV_ENC28J60_RegisterStatisticsGet Function

Get the register statistics.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Get Register Statistics

Get the device specific statistics.

Remarks

Statistics are not planned for the first release

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

pRegEntries	
nEntries	
pHwEntries	

DRV_ENC28J60_StatisticsGet Function

Retrieve the devices statistics.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Get Statistics

Get the current statistics stored in the driver.

Remarks

Statistics are not planned for the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Status Function

Gets the current status of the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_STATUS DRV_ENC28J60_Status(SYS_MODULE_OBJ obect);
```

Returns

- SYS_STATUS_ERROR - if an invalid handle has been passed in
- SYS_STATUS_UNINITIALIZED - if the driver has not completed initialization
- SYS_STATUS_BUSY - if the driver is closing and moving to the closed state
- SYS_STATUS_READY - if the driver is ready for client commands

Description

ENC28J60 Status

This function will get the status of the driver instance.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize().

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENC28J60_Initialize()

c) Receive Functions

DRV_ENC28J60_PacketRx Function

Receive a packet from the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_PACKET* DRV_ENC28J60_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- Pointer to a valid packet - if successful
- NULL - if unsuccessful

Description

ENC28J60 Receive Packet

This function retrieves a packet from the driver. The packet needs to be acknowledged with the linked acknowledge function so it can be reused.

Remarks

ppPktStat is ignored in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRes	the result of the operation
ppPktStat	pointer to the receive statistics

DRV_ENC28J60_RxFilterHashTableEntrySet Function

This function adds an entry to the hash table.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_RxFilterHashTableEntrySet(DRV_HANDLE hMac, const TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Receive Filter Hash Table Entry Set

This function adds to the MAC's hash table for hash table matching.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
DestMACAddr	MAC address to add to the hash table

d) Transmit Functions

DRV_ENC28J60_PacketTx Function

This function queues a packet for transmission.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OP_ERR - if the client handle is invalid
- TCPIP_MAC_RES_IS_BUSY - if the driver is not in the run state
- TCPIP_MAC_RES_QUEUE_TX_FULL - if there are no free descriptors
- TCPIP_MAC_RES_OK - on successful queuing of the packet

Description

ENC28J60 Packet Transmit

This function will take a packet and add it to the queue for transmission. When the packet has finished transmitting the driver will call the packets acknowledge function. When that acknowledge function is complete the driver will forget about the packet.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
ptrPacket	pointer to the packet

e) Event Functions

DRV_ENC28J60_EventAcknowledge Function

Acknowledges an event.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

Returns

- true - if successful
- false - if not successful

Description

ENC28J60 Acknowledge Event

This function acknowledges an event.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the events to acknowledge

DRV_ENC28J60_EventMaskSet Function

Sets the event mask.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if the mask could be set
- false - if the mast could not be set

Description

ENC28J60 Set Event Mask

Sets the event mask to what is passed in.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the mask to enable or disable
enable	to enable or disable events

DRV_ENC28J60_EventPendingGet Function

Gets the current events.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_EVENT DRV_ENC28J60_EventPendingGet(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_EV_NONE - Returned on an error
- List of events - Returned on event other than an error

Description

ENC28J60 Get Events

This function gets the current events.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

f) Data Types and Constants

DRV_ENC28J60_Configuration Structure

Defines the data required to initialize or reinitialize the ENC28J60 Driver.

File

drv_enc28j60.h

C

```
typedef struct _DRV_ENC28J60_Configuration {
  uint16_t txDescriptors;
  uint16_t rxDescriptors;
  uint16_t rxDescBufferSize;
  SYS_MODULE_INDEX spiDrvIndex;
  uint32_t spiBps;
  uint16_t rxBufferSize;
  uint16_t maxFrameSize;
  PORTS_MODULE_ID spiSSPortModule;
  PORTS_CHANNEL spiSSPortChannel;
  PORTS_BIT_POS spiSSPortPin;
  bool intEnable;
  PORTS_MODULE_ID intPortModule;
  PORTS_CHANNEL intPortChannel;
  PORTS_BIT_POS intPortPin;
  DRV_ENC28J60_MDIX_TYPE mdixControl;
  PORTS_MODULE_ID mdixPortModule;
  PORTS_CHANNEL mdixPortChannel;
  PORTS_BIT_POS mdixPortPin;
} DRV_ENC28J60_Configuration;
```

Members

Members	Description
uint16_t txDescriptors;	Number of TX Descriptors to Allocate
uint16_t rxDescriptors;	Number of RX Descriptors to Allocate
uint16_t rxDescBufferSize;	Size of the buffer each RX Descriptor will use. Make sure its not smaller than maxFrameSize
SYS_MODULE_INDEX spiDrvIndex;	Index of the SPI driver to use
uint32_t spiBps;	Bus speed to use for the SPI interface. Section 1.0 of the ENC28J60 data sheets says the maximum is 20000000 Hz. It is not recommended to go above this value.
uint16_t rxBufferSize;	The ENC28J60 hardware has a 8 k dram. rxBufferSize defines how much of that memory is used by the rxBuffer
uint16_t maxFrameSize;	The maximum frame size to be supported by the hardware. 1536 is the default
PORTS_MODULE_ID spiSSPortModule;	Port Module of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
PORTS_CHANNEL spiSSPortChannel;	Port Channel of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
PORTS_BIT_POS spiSSPortPin;	Pin position of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
bool intEnable;	Use Interrupts or not.
PORTS_MODULE_ID intPortModule;	Port Module of the GPIO pin hooked up to the INT pin of the ENC28J60
PORTS_CHANNEL intPortChannel;	Port Channel of the GPIO pin hooked up to the INT pin of the ENC28J60
PORTS_BIT_POS intPortPin;	Pin Position of the GPIO pin hooked up to the INT pin of the ENC28J60

DRV_ENC28J60_MDIX_TYPE mdixControl;	To select the control type of the MDIX. This is only needed for hooking up to switches that don't have auto-mdix.
PORTS_MODULE_ID mdixPortModule;	Port Module of the GPIO pin hooked up to the MDIX select pin
PORTS_CHANNEL mdixPortChannel;	Port Channel of the GPIO pin hooked up to the MDIX select pin
PORTS_BIT_POS mdixPortPin;	Pin Position of the GPIO pin hooked up to the MDIX select pin

Description

ENC28J60 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the ENC28J60 driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_ENC28J60_MDIX_TYPE Enumeration

Defines the enumeration for controlling the MDIX select.

File

drv_enc28j60.h

C

```
typedef enum {
  DRV_ENC28J60_NO_CONTROL = 0,
  DRV_ENC28J60_NORMAL,
  DRV_ENC28J60_REVERSE = 0
} DRV_ENC28J60_MDIX_TYPE;
```

Members

Members	Description
DRV_ENC28J60_NO_CONTROL = 0	No Control
DRV_ENC28J60_NORMAL	Normal MDIX
DRV_ENC28J60_REVERSE = 0	Reverse MDIX

Description

ENC28J60 Driver MDIX Control type

This type defines the enumeration for controlling the MDIX select.

Remarks

None.

DRV_ENC28J60_MACObject Variable

File

drv_enc28j60.h

C

```
const TCPIP_MAC_OBJECT DRV_ENC28J60_MACObject;
```

Description

ENC28J60 External MAC Virtualization Table

Files

Files

Name	Description
drv_enc28j60.h	ENC28J60 Driver interface definition.
drv_enc28j60_config_template.h	enc28j60 Driver configuration definitions template.

Description

drv_enc28j60.h

ENC28J60 Driver interface definition.

Enumerations

	Name	Description
	DRV_ENC28J60_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Functions

	Name	Description
	DRV_ENC28J60_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENC28J60_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENC28J60_Deinitialize	Deinitializes the ENC28J60 Driver Instance. **Implementation:** Dynamic
	DRV_ENC28J60_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENC28J60_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENC28J60_EventPendingGet	Gets the current events. **Implementation:** Dynamic
	DRV_ENC28J60_Initialize	Initializes the ENC28J60 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENC28J60_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENC28J60_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENC28J60_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENC28J60_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic
	DRV_ENC28J60_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENC28J60_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENC28J60_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENC28J60_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Reinitialize	Reinitializes the instance of the ENC28J60 driver. **Implementation:** Dynamic
	DRV_ENC28J60_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic
	DRV_ENC28J60_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENC28J60_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENC28J60_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Status	Gets the current status of the driver. **Implementation:** Dynamic
	DRV_ENC28J60_Tasks	Main task function for the driver. **Implementation:** Dynamic

Structures

	Name	Description
	_DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.

Variables

	Name	Description
	DRV_ENC28J60_MACObject	ENC28J60 External MAC Virtualization Table

Description

ENC28J60 Driver Public Interface

This file defines the interface definition for the ENC28J60 Driver.

File Name

drv_enc28j60.h

Company

Microchip Technology Inc.

drv_enc28j60_config_template.h

enc28j60 Driver configuration definitions template.

Macros

	Name	Description
	DRV_ENC28J60_CLIENT_INSTANCES	Selects the maximum number of clients.
	DRV_ENC28J60_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

enc28j60 Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_enc28j60_config_template.h

Company

Microchip Technology Inc.

ENCx24J600 Driver Library Help

This section provides information on the ENCx24J600 Driver Library.

Introduction

This library provides a driver-level abstraction of the ENCx24J600 Ethernet MAC that can be connected to the PIC32. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module for details.

The "Host-To-Network"_layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section.

Description

The ENCx24J600 External MAC is an external module to the PIC32 that is connected through a SPI or PSP interface. This driver interfaces with the SPI driver to communicate with the external device to implement a complete Ethernet node in a system.

The following are some of the key features of this module:

- Supports 10/100 Ethernet
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Manual and automatic flow control
- Supports Auto-MDIX

- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Runt Packet Detection and Filtering
- Supports Packet Payload Checksum calculation
- CRC Check
- Supports SPI interface

Using the Library

This topic describes the basic architecture and functionality of the Ethernet MAC driver and is meant for advanced users or TCP/IP stack driver developers.

Description

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system-wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

Interface Header File: `drv_encx24j600.h`

The interface to the ENCx24J600 Driver Library is defined in the `drv_encx24j600.h` header file. Any C language source (`.c`) file that uses the ENCx24J600 Driver Library should include `drv_encx24j600.h`.

Library File: The ENCx24J600 Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The ENCx24J600 Driver Library provides the low-level abstraction of the communications protocol to communicate to the ENCx24J600 external MAC though the SPI peripheral on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the ENCx24J600 Driver Library interface.

Description

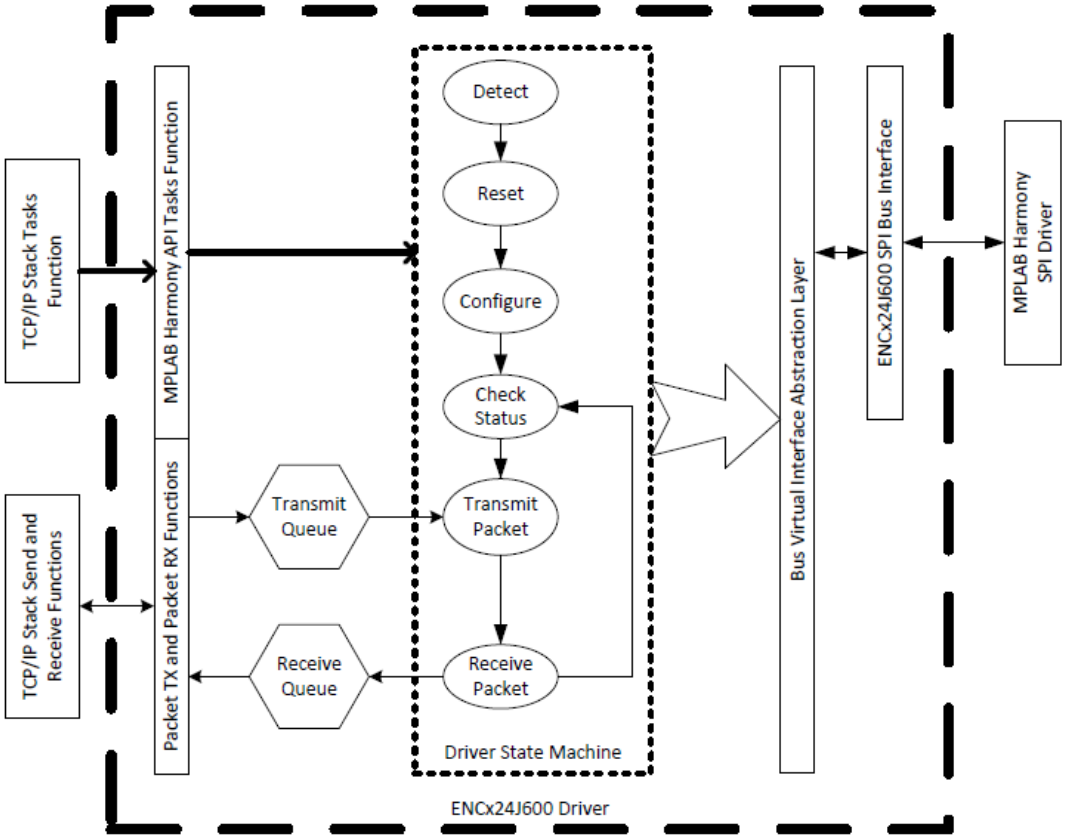
The ENCx24J600 Driver library has several different layers to it, as illustrated in the following figure. The interface layer has two main sections that are used the most often: The Tasks function, and the TCP/IP Send and Receive functions.

The Tasks function manages the internal state machine which detects, resets, and then configures the ENCx24J600 External MAC. It also handles the monitoring of the hardware status, sending and receiving packets.

The TCP/IP Send and Receive functions interact with the RAM-based queue of packets that are queued to send and packets that have been queued waiting for pick-up by the stack.

The main state machine does not interface directly to the SPI bus, but instead, interfaces to a virtual bus abstraction layer that allows for the replacement of the specific underlying bus implementation.

Abstraction Model



Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the ENCx24J600 Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Status Functions	Provides status functions.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support the TCP/IP virtual MAC interface.

Configuring the SPI Driver

This section describes the configuration settings for the ENCx24J600 Driver Library.

Description

Configuration

The ENC hardware requires a specific configuration of the SPI driver to work correctly. Inside the MHC SPI Driver configuration be sure to select:

- SPI clock rate of 14000000 or less. With a PB clock of 80 MHz, 13333333 is the clock rate.
- Clock mode of DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
- Input phase of SPI_INPUT_SAMPLING_PHASE_AT_END

Recommended Settings

- Interrupt Driver mode
- Enhanced Buffer mode
- DMA mode enabled:
- DMA block transfer size of at least 1600 bytes
- Size of DMA buffer for dummy data of at least 1600 bytes
- Ensure when setting up DMA in interrupt mode that the DMA interrupts are a higher priority than the SPI Driver interrupt

Example:

```
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER           1
#define DRV_SPI_CLIENTS_NUMBER             1
#define DRV_SPI_ELEMENTS_PER_QUEUE         30

/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE             2048
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE      2048

/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0              SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0           DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0            DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0      false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0          SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0           CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0           13333333
#define DRV_SPI_BUFFER_TYPE_IDX0         DRV_SPI_BUFFER_TYPE_ENHANCED
#define DRV_SPI_CLOCK_MODE_IDX0          DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0         SPI_INPUT_SAMPLING_PHASE_AT_END
#define DRV_SPI_TX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0          INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0        INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0    INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0          30
#define DRV_SPI_RESERVED_JOB_IDX0        1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0      DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0    16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0      DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0    16 Driver Library
```

Configuring the Library

The configuration of the ENCx24J600 Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the ENCX24J600 Driver Library. Based on the selections made, the ENCx24J600 Driver Library may support the selected features. These configuration settings will apply to all instances of the ENCx24J600 Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the ENCx24J600 Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/encx24j600`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source Folder Name	Description
/drv_encx24j600.h	This file provides the interface definitions of the ENCx24J600 Driver.

Required File(s)

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

Source Folder Name	Description
/src/dynamic/drv_drv_encx24J600_api.c	This file contains the API function implementations.
/src/dynamic/drv_encx24J600_main_state.c	This file contains the main state machine functions.
/src/dynamic/drv_encx24J600_utils.c	This file contains functions that are used throughout the driver.
/src/dynamic/bus/spi/drv_encx24J600_spi_bus.c	This file contains the functions to interface with the SPI bus.
/src/dynamic/closed_state/drv_encx24J600_closed_state.c	This file contains the functions for handling the driver closed state.
/src/dynamic/initialization_state/drv_encx24J600_configure_state.c	This file contains the functions for configuring the ENC hardware.
/src/dynamic/initialization_state/drv_encx24J600_detect_state.c	This file contains the functions for detecting the ENC hardware.
/src/dynamic/initialization_state/drv_encx24J600_initialization_state.c	This file contains the functions for the initialization state machine.
/src/dynamic/initialization_state/drv_encx24J600_reset_state.c	This file contains the functions for resetting the ENC hardware.
/src/dynamic/packet/drv_encx24J600_rx_packet.c	This file contains the functions for receiving a packet from the ENC hardware.
/src/dynamic/packet/drv_encx24J600_tx_packet.c	This file contains the functions for sending a packet to the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_change_duplex_state.c	This file contains the functions for configuring the duplex mode of the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_check_int_state.c	This file contains the functions for checking and processing the ENC hardware interrupts.
/src/dynamic/running_state/drv_encx24J600_check_status_state.c	This file contains the functions for checking the status of the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_check_tx_status_state.c	This file contains the functions for checking the status of a transmitted packet.
/src/dynamic/running_state/drv_encx24J600_running_state.c	This file contains the functions for managing the running state machine.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source Folder Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The ENCx24J600 Driver Library depends on the following modules:

- SPI Driver Library
- TCP/IP Stack Library
- TCP/IP Stack MAC Driver Module

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_ENCX24J600_Deinitialize	Deinitializes the ENCx24J600 Driver Instance. **Implementation:** Dynamic
	DRV_ENCX24J600_Initialize	Initializes the ENCx24J600 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENCX24J600_Reinitialize	Reinitializes the instance of the ENCX24J600 driver. **Implementation:** Dynamic
	DRV_ENCX24J600_Tasks	Main task function for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENCX24J600_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ENCX24J600_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENCX24J600_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENCX24J600_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENCX24J600_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Status	Gets the current status of the driver. **Implementation:** Dynamic

c) Receive Functions

	Name	Description
	DRV_ENCX24J600_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic

d) Transmit Functions

	Name	Description
	DRV_ENCX24J600_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic

e) Event Functions

	Name	Description
	DRV_ENCX24J600_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic

	DRV_ENCX24J600_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENCX24J600_EventPendingGet	Gets the current events. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	_DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Description

This section describes the Application Programming Interface (API) functions of the ENCx24J600 Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_ENCX24J600_Deinitialize Function

Deinitializes the ENCx24J600 Driver Instance.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENCX24J600 Deinitialization

This function deallocates any resources allocated by the initialization function.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
Object	the valid object returned from DRV_ENCX24J600_Initialize

DRV_ENCX24J600_Initialize Function

Initializes the ENCx24J600 Driver Instance, with the configuration data.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_MODULE_OBJ DRV_ENCX24J600_Initialize(SYS_MODULE_INDEX index, SYS_MODULE_INIT * init);
```

Returns

- Valid handle to the driver instance - If successful
- SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENCX24J600 Initialization

This function initializes the ENCx24J600 Driver with configuration data passed into it by either the system_init function or by the DRV_ENCX24J600_StackInitialize function. Calling this function alone is not enough to initialize the driver, DRV_ENCX24J600_SetMacCtrlInfo must be called with valid data before the driver is ready to be opened.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a DRV_ENX24J600_CONFIG structure.

DRV_ENCX24J600_Reinitialize Function

Reinitializes the instance of the ENCX24J600 driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

ENCX24J600 Reinitialization

This function will deinitialize and initialize the driver instance. As with DRV_ENCX24J600_Initialize DRV_ENCX24J600_SetMacCtrlInfo must be called for the driver to be useful.

Remarks

This function is not planned to be implemented for the first release.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

DRV_ENCX24J600_Tasks Function

Main task function for the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENCX24J600 Tasks

This function will execute the main state machine for the ENCX24J600 driver.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENCX24J600_Initialize

DRV_ENCX24J600_SetMacCtrlInfo Function

This function sets the MAC control information for the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_SetMacCtrlInfo(SYS_MODULE_OBJ object, TCPIP_MAC_MODULE_CTRL * init);
```

Returns

None.

Description

ENCX24J600 Set MAC Control Information

This function is used to pass in the TCPIP_MAC_CONTROL_INIT information that is used for allocation and deallocation of memory, event signaling, etc. This function is needed to be called so that the driver can enter initialization state when the tasks function is called.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

DRV_ENCX24J600_StackInitialize Function

This function initializes the driver with a TCPIP_MAC_INIT object.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_MODULE_OBJ DRV_ENCX24J600_StackInitialize(SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

Returns a valid handle to the driver instance - If successful SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENCX24J600 Stack Initialization

This function is used by the TCP/IP stack to fully initialize the driver with both the ENCX24J600 specific configuration and the MAC control information. With this function there is no need to call DRV_ENCX24J600_SetMacCtrlInfo.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a TCPIP_MAC_INIT structure.

DRV_ENCX24J600_Process Function

Additional processing that happens outside the tasks function.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Process

This function does additional processing that is not done inside the tasks function.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

b) Client Level Functions

DRV_ENCX24J600_Close Function

Closes a client handle to the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Close(DRV_HANDLE handle);
```

Returns

None.

Description

ENCX24J600 Close

This function closes a handle to the driver. If it is the last client open, the driver will send an RX Disable command to the ENC hardware and move to the closed state.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
handle	The successfully opened handle

DRV_ENCX24J600_ConfigGet Function

Gets the current configuration.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
size_t DRV_ENCX24J600_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

Number of bytes copied to the buffer

Description

ENCX24J600 Get Configuration

Gets the current configuration.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
configBuff	location to copy the configuration too
buffSize	buffer size
pConfigSize	configuration size needed

DRV_ENCX24J600_LinkCheck Function

This function returns the status of the link.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is active
- false - all other times

Description

ENCX24J600 Link Check

This function checks the status of the link and returns it to the caller.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENCX24J600_Open Function

This function is called by the client to open a handle to a driver instance.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
DRV_HANDLE DRV_ENCX24J600_Open(SYS_MODULE_INDEX index, DRV_IO_INTENT intent);
```

Returns

Returns a valid handle - If successful INVALID_HANDLE - If unsuccessful

Description

ENCX24J600 Open

The client will call this function to open a handle to the driver. When the first instance is opened than the driver will send the RX enabled command to the ENC hardware.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
intent	The intent to use when opening the driver. Only exclusive is supported

DRV_ENCX24J600_ParametersGet Function

Get the parameters of the device.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OK - if the hMac is valid

Description

ENCX24J600 Get Parameters

Get the parameters of the device, which includes that it is an Ethernet device and what it's MAC address is.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pMacParams	pointer to put the parameters

DRV_ENCX24J600_PowerMode Function

This function sets the power mode of the device.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_PowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- false - This functionality is not supported in this version of the driver

Description

ENCX24J600 Power Mode

This function sets the power mode of the ENCX24J600.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pwrMode	the power mode to set

DRV_ENCX24J600_RegisterStatisticsGet Function

Get the register statistics.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Get Register Statistics

Get the device specific statistics.

Remarks

Statistics are not planned for the first release

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRegEntries	
nEntries	
pHwEntries	

DRV_ENCX24J600_StatisticsGet Function

Retrieve the devices statistics.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Get Statistics

Get the current statistics stored in the driver.

Remarks

Statistics are not planned for the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENCX24J600_Status Function

Gets the current status of the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_STATUS DRV_ENCX24J600_Status(SYS_MODULE_OBJ obect);
```

Returns

- SYS_STATUS_ERROR - if an invalid handle has been passed in
- SYS_STATUS_UNINITIALIZED - if the driver has not completed initialization
- SYS_STATUS_BUSY - if the driver is closing and moving to the closed state
- SYS_STATUS_READY - if the driver is ready for client commands

Description

ENCX24J600 Status

This function will get the status of the driver instance.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize().

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENCX24J600_Initialize()

c) Receive Functions

DRV_ENCX24J600_PacketRx Function

Receive a packet from the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_PACKET* DRV_ENCX24J600_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- Pointer to a valid packet - if successful
- NULL - if unsuccessful

Description

ENCX24J600 Receive Packet

This function retrieves a packet from the driver. The packet needs to be acknowledged with the linked acknowledge function so it can be reused.

Remarks

ppPktStat is ignored in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRes	the result of the operation
ppPktStat	pointer to the receive statistics

DRV_ENCX24J600_RxFilterHashTableEntrySet Function

This function adds an entry to the hash table.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_RxFilterHashTableEntrySet(DRV_HANDLE hMac, const TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Receive Filter Hash Table Entry Set

This function adds to the MAC's hash table for hash table matching.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
DestMACAddr	MAC address to add to the hash table

d) Transmit Functions

DRV_ENCX24J600_PacketTx Function

This function queues a packet for transmission.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OP_ERR - if the client handle is invalid
- TCPIP_MAC_RES_IS_BUSY - if the driver is not in the run state
- TCPIP_MAC_RES_QUEUE_TX_FULL - if there are no free descriptors
- TCPIP_MAC_RES_OK - on successful queuing of the packet

Description

ENCX24J600 Packet Transmit

This function will take a packet and add it to the queue for transmission. When the packet has finished transmitting the driver will call the packets acknowledge function. When that acknowledge function is complete the driver will forget about the packet.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
ptrPacket	pointer to the packet

e) Event Functions

DRV_ENCX24J600_EventAcknowledge Function

Acknowledges an event.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

Returns

- true - if successful
- false - if not successful

Description

ENCX24J600 Acknowledge Event

This function acknowledges an event.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the events to acknowledge

DRV_ENCX24J600_EventMaskSet Function

Sets the event mask.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if the mask could be set
- false - if the mast could not be set

Description

ENCX24J600 Set Event Mask

Sets the event mask to what is passed in.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the mask to enable or disable
enable	to enable or disable events

DRV_ENCX24J600_EventPendingGet Function

Gets the current events.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_EVENT DRV_ENCX24J600_EventPendingGet(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_EV_NONE - Returned on an error
- List of events - Returned on event other than an error

Description

ENCX24J600 Get Events

This function gets the current events.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

f) Data Types and Constants

DRV_ENCX24J600_Configuration Structure

Defines the data required to initialize or reinitialize the ENCX24J600 Driver.

File

drv_encx24j600.h

C

```
typedef struct _DRV_ENCX24J600_Configuration {
  uint16_t txDescriptors;
  uint16_t rxDescriptors;
  uint16_t rxDescBufferSize;
  SYS_MODULE_INDEX spiDrvIndex;
  uint32_t spiBps;
  uint16_t rxBufferSize;
  uint16_t maxFrameSize;
  PORTS_MODULE_ID spiSSPortModule;
  PORTS_CHANNEL spiSSPortChannel;
  PORTS_BIT_POS spiSSPortPin;
  bool intEnable;
  PORTS_MODULE_ID intPortModule;
  PORTS_CHANNEL intPortChannel;
  PORTS_BIT_POS intPortPin;
  DRV_ENCX24J600_MDIX_TYPE mdixControl;
  PORTS_MODULE_ID mdixPortModule;
  PORTS_CHANNEL mdixPortChannel;
  PORTS_BIT_POS mdixPortPin;
  TCPIP_ETH_OPEN_FLAGS ethType;
  TCPIP_ETH_OPEN_FLAGS dupMode;
} DRV_ENCX24J600_Configuration;
```

Members

Members	Description
uint16_t txDescriptors;	Number of TX Descriptors to Allocate
uint16_t rxDescriptors;	Number of RX Descriptors to Allocate
uint16_t rxDescBufferSize;	Size of the buffer each RX Descriptor will use. Make sure its not smaller that maxFrameSize
SYS_MODULE_INDEX spiDrvIndex;	Index of the SPI driver to use
uint32_t spiBps;	Bus speed to use for the SPI interface. Section 1.0 of the ENCX24J600 data sheets says the maximum is 14000000 Hz. It is not recommended to go above this value.
uint16_t rxBufferSize;	The ENCX24J600 hardware has a 22 k dram. rxBufferSize defines how much of that memory is used by the rxBuffer
uint16_t maxFrameSize;	The maximum frame size to be supported by the hardware. 1536 is the default
PORTS_MODULE_ID spiSSPortModule;	Port Module of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
PORTS_CHANNEL spiSSPortChannel;	Port Channel of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
PORTS_BIT_POS spiSSPortPin;	Pin position of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
bool intEnable;	Use Interrupts or not.
PORTS_MODULE_ID intPortModule;	Port Module of the GPIO pin hooked up to the INT pin of the ENCX24J600
PORTS_CHANNEL intPortChannel;	Port Channel of the GPIO pin hooked up to the INT pin of the ENCX24J600
PORTS_BIT_POS intPortPin;	Pin Position of the GPIO pin hooked up to the INT pin of the ENCX24J600
DRV_ENCX24J600_MDIX_TYPE mdixControl;	To select the control type of the MDIX. This is only needed for hooking up to switches that don't have auto-mdix.
PORTS_MODULE_ID mdixPortModule;	Port Module of the GPIO pin hooked up to the MDIX select pin
PORTS_CHANNEL mdixPortChannel;	Port Channel of the GPIO pin hooked up to the MDIX select pin
PORTS_BIT_POS mdixPortPin;	Pin Position of the GPIO pin hooked up to the MDIX select pin
TCPIP_ETH_OPEN_FLAGS ethType;	Ethernet type
TCPIP_ETH_OPEN_FLAGS dupMode;	Duplex Mode

Description

ENCX24J600 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the ENCX24J600 driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_ENCX24J600_MDIX_TYPE Enumeration

Defines the enumeration for controlling the MDIX select.

File

drv_encx24j600.h

C

```
typedef enum {
  DRV_ENCX24J600_NO_CONTROL = 0,
  DRV_ENCX24J600_NORMAL,
  DRV_ENCX24J600_REVERSE = 0
} DRV_ENCX24J600_MDIX_TYPE;
```

Members

Members	Description
DRV_ENCX24J600_NO_CONTROL = 0	No Control
DRV_ENCX24J600_NORMAL	Normal MDIX
DRV_ENCX24J600_REVERSE = 0	Reverse MDIX

Description

ENCX24J600 Driver MDIX Control type

This type defines the enumeration for controlling the MDIX select.

Remarks

None.

Files

Files

Name	Description
drv_encx24j600.h	ENCx24J600 Driver interface definition.

Description

drv_encx24j600.h

ENCx24J600 Driver interface definition.

Enumerations

	Name	Description
	DRV_ENCX24J600_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Functions

	Name	Description
	DRV_ENCX24J600_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENCX24J600_Deinitialize	Deinitializes the ENCx24J600 Driver Instance. **Implementation:** Dynamic
	DRV_ENCX24J600_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic

	DRV_ENCX24J600_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENCX24J600_EventPendingGet	Gets the current events. **Implementation:** Dynamic
	DRV_ENCX24J600_Initialize	Initializes the ENCx24J600 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENCX24J600_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENCX24J600_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENCX24J600_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic
	DRV_ENCX24J600_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENCX24J600_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Reinitialize	Reinitializes the instance of the ENCX24J600 driver. **Implementation:** Dynamic
	DRV_ENCX24J600_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic
	DRV_ENCX24J600_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENCX24J600_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Status	Gets the current status of the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_Tasks	Main task function for the driver. **Implementation:** Dynamic

Structures

	Name	Description
	_DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.

Description

ENCx24J600 Driver Public Interface

This file defines the interface definition for the ENCx24J600 Driver.

File Name

drv_encx24j600.h

Company

Microchip Technology Inc.

Ethernet MAC Driver Library

This section describes the Ethernet MAC Driver Library.

Introduction

This library provides a driver-level abstraction of the on-chip Ethernet Controller found on many PIC32 devices. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module help for details.

The "Host-To-Network" layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section. An external Ethernet "PHY" provides the Physical layer, providing conversion between the digital and analog.

Description

The PIC32 Ethernet Controller is a bus master module that interfaces with an off-chip PHY to implement a complete Ethernet node in a system. The following are some of the key features of this module:

- Supports 10/100 Ethernet
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Manual and automatic flow control
- Supports Auto-MDIX enabled PHYs
- Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY data interfaces
- Performance statistics metrics in hardware.
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Runt Packet Detection and Filtering
- Supports Packet Payload Checksum calculation
- CRC Check

Support for the Serial Management Interface (SMI) (also known as the MIIM interface) is provided by the Ethernet PHY Driver Library.

Using the Library

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

This topic describes the basic architecture and functionality of the Ethernet MAC driver and is meant for advanced users or TCP/IP stack driver developers.

Interface Header File: `drv_ethmac.h`

The interface to the Ethernet MAC library is defined in the `drv_ethmac.h` header file, which is included by the MPLAB Harmony TCP/IP stack.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the Ethernet MAC Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

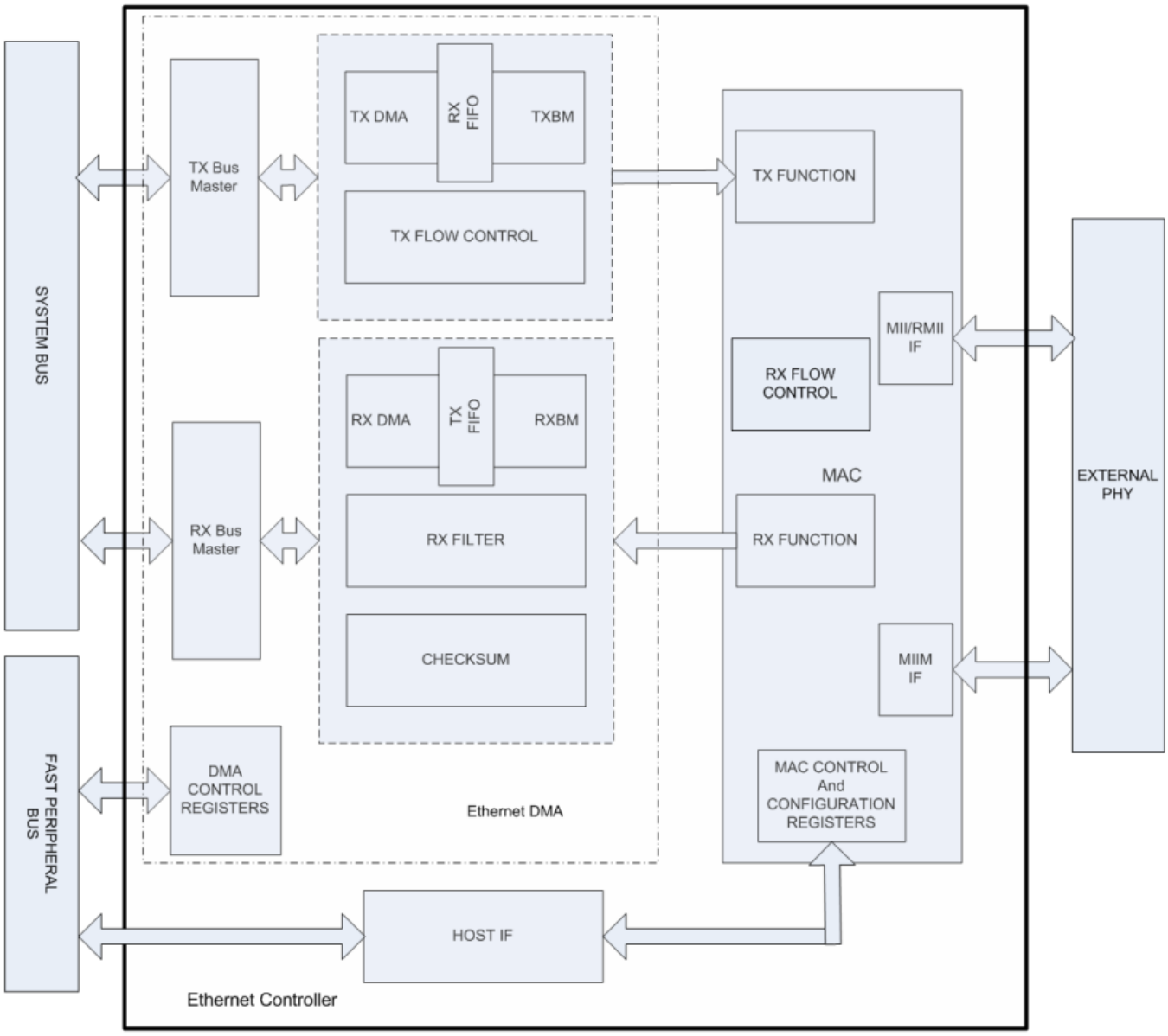
Description

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external Ethernet PHY chip. The PHY chip provides a digital-analog interface as part of the Physical Layer and the controller provides the Media Access Controller (MAC) layer above the PHY.

As shown in Figure 1, the Ethernet Controller consists of the following modules:

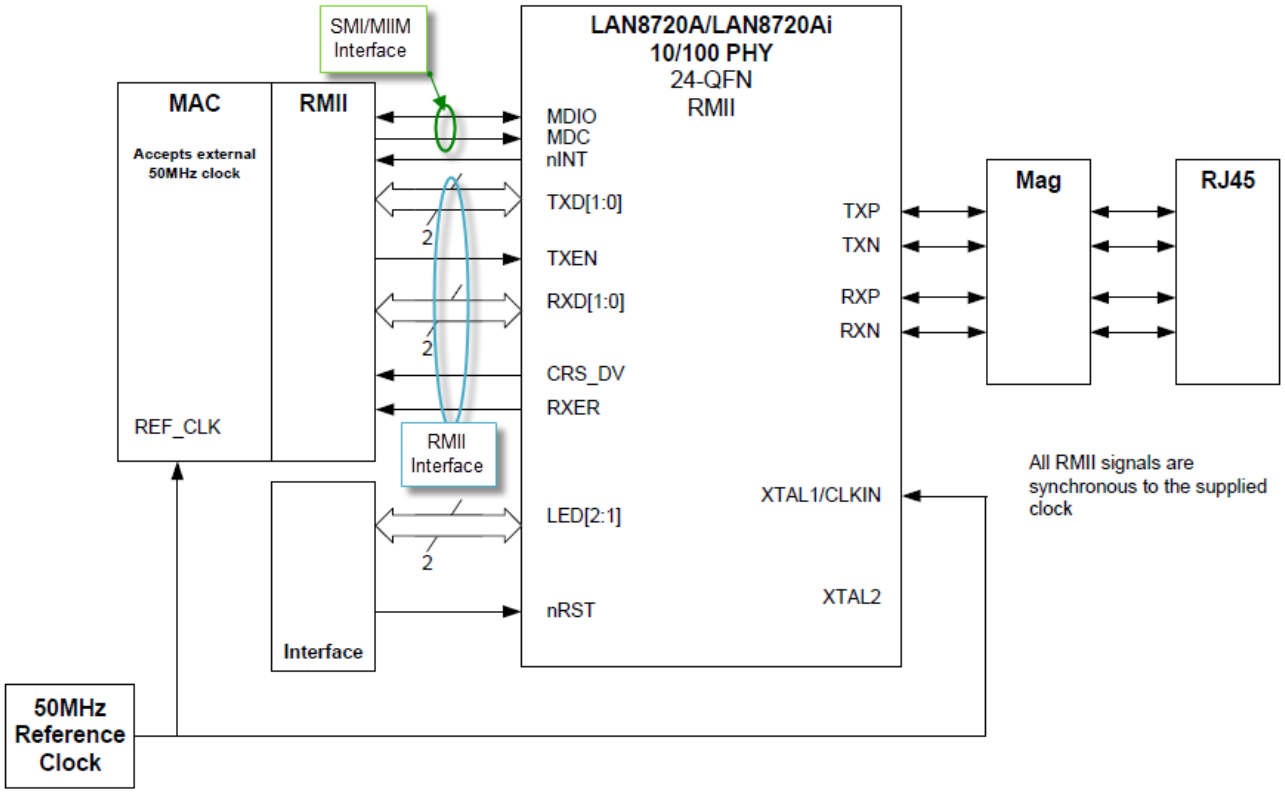
- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block: Responsible for control of the transmission of PAUSE frames. (Reception of PAUSE frames is handled within the MAC.)
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected
- TX DMA/TX Buffer Management Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX Buffer Management Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Figure 1: Ethernet Controller Block Diagram



For completeness, we also need to look at the interface diagram of a representative Ethernet PHY. As shown in Figure 2, the PHY has two interfaces, one for configuring and managing the PHY (SMI/MIIM) and another for transmit and receive data (RMII or MII). The SMI/MIIM interface is the responsibility of the Ethernet PHY Driver Library. When setting up the Ethernet PHY, this Ethernet driver calls primitives from the Ethernet PHY Driver library. The RMII/MII data interface is the responsibility of the Ethernet MAC Driver Library (this library).

Figure 2: Ethernet PHY Interfaces



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system. Refer to the TCP/IP Stack Library MAC Driver Module help for the interface that the Ethernet driver has to implement in a MPLAB Harmony system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet MAC Driver Library.

Library Interface Section	Description
Client Level Functions	DRV_ETHMAC_PIC32MACOpen, DRV_ETHMAC_PIC32MACClose, and DRV_ETHMAC_PIC32MACSetup to support the TCP/IP Stack. Plus link status and power options.
Receive Functions	Receive routines.
Transmit Functions	Transmit routines.
Event Functions	Ethernet event support routines.
Other Functions	Additional routines.
Data Types and Constants	Typedefs and #defines.

Configuring the Library

Macros

	Name	Description
	DRV_ETHMAC_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHMAC_INDEX	Ethernet MAC static index selection.
	DRV_ETHMAC_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHMAC_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ETHMAC_INTERRUPT_SOURCE	Defines an override of the interrupt source in case of static driver.
	DRV_ETHMAC_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHMAC_POWER_STATE	Defines an override of the power state of the Ethernet MAC driver.

Description

The configuration of the Ethernet MAC driver is done as part of the MPLAB Harmony TCP/IP Stack configuration and is based on the `system_config.h` file, which may include the `tcpip_mac_config.h`. See the TCP/IP Stack Library MAC Driver Module help file for configuration options.

This header file contains the configuration selection for the Ethernet MAC Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ETHMAC_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_CLIENTS_NUMBER 1
```

Description

Ethernet MAC Maximum Number of Clients

This definition select the maximum number of clients that the Ethernet MAC driver can support at run time.

Remarks

The MAC driver is not a true multi-client driver. Under normal usage, the only client of the MAC driver is the TCP/IP stack. After the MAC driver provided an DRV_HANDLE as a result of an Open operation, any other attempt to call Open will return a invalid handle. Default value should be 1.

However, for allowing other modules to interface directly with the MAC driver while the TCP/IP stack currently uses the the MAC driver this symbol can have a value greater than 1. But the returned handle is the same one as the TCP/IP stack uses.

DRV_ETHMAC_INDEX Macro

Ethernet MAC static index selection.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INDEX DRV_ETHMAC_INDEX_1
```

Description

Ethernet MAC Static Index Selection

This definition selects the Ethernet MAC static index for the driver object reference

Remarks

This index is required to make a reference to the driver object.

DRV_ETHMAC_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INSTANCES_NUMBER 1
```

Description

Ethernet MAC hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

DRV_ETHMAC_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INTERRUPT_MODE true
```

Description

Ethernet MAC Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_ETHMAC_INTERRUPT_SOURCE Macro

Defines an override of the interrupt source in case of static driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INTERRUPT_SOURCE INT_SOURCE_ETH_1
```

Description

Ethernet MAC Interrupt Source

Defines an override of the interrupt source in case of static driver.

Remarks

Refer to the INT PLIB document for more information on INT_SOURCE enumeration.

DRV_ETHMAC_PERIPHERAL_ID Macro

Defines an override of the peripheral ID.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_PERIPHERAL_ID ETHMAC_ID_1
```

Description

Ethernet MAC Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHMAC_ID_0

DRV_ETHMAC_POWER_STATE Macro

Defines an override of the power state of the Ethernet MAC driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

Ethernet MAC power state configuration

Defines an override of the power state of the Ethernet MAC driver.

Remarks

This feature may not be available in the device or the Ethernet MAC module selected.

Building the Library

This section lists the files that are available in the Ethernet MAC Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/ethmac`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ethmac.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ethmac.c`	PIC32 internal Ethernet driver virtual MAC implementation file.
`/src/dynamic/drv_ethmac_lib.c`	PIC32 internal Ethernet driver controller implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Ethernet MAC Driver Library depends on the following modules:

- Ethernet PHY Driver Library
- Interrupt System Service Library
- Timer System Service Library
- Ethernet Peripheral Library

Library Interface

a) Client Level Functions

	Name	Description
	DRV_ETHMAC_PIC32MACClose	Closes a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic

	DRV_ETHMAC_PIC32MACDeinitialize	Deinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACInitialize	Initializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACLinkCheck	Checks current link status. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACOpen	Opens a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACParametersGet	MAC parameter get function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPowerMode	Selects the current power mode for the Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACProcess	MAC periodic processing function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatisticsGet	Gets the current MAC statistics. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatus	Provides the current status of the MAC driver module. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACConfigGet	Gets the current MAC driver configuration. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRegisterStatisticsGet	Gets the current MAC hardware statistics registers. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACReinitialize	Reinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic

b) Receive Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPacketRx	This is the MAC receive function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet	Sets the current MAC hash table receive filter. **Implementation:** Dynamic

c) Transmit Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPacketTx	MAC driver transmit function. **Implementation:** Dynamic

d) Event Functions

	Name	Description
	DRV_ETHMAC_PIC32MACEventAcknowledge	Acknowledges and re-enables processed events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventMaskSet	Enables/disables the MAC events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventPendingGet	Returns the currently pending events. **Implementation:** Dynamic

e) Other Functions

	Name	Description
	DRV_ETHMAC_Tasks_ISR	Ethernet MAC driver interrupt function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACTasks	Maintains the EThernet MAC driver's state machine. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_ETHMAC_INDEX_1	This is macro DRV_ETHMAC_INDEX_1.
	DRV_ETHMAC_INDEX_0	Ethernet driver index definitions.
	DRV_ETHMAC_INDEX_COUNT	Number of valid Ethernet driver indices.

Description

This section lists the interface routines, data types, constants and macros for the library.

a) Client Level Functions

DRV_ETHMAC_PIC32MACClose Function

Closes a client instance of the PIC32 MAC Driver.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACClose(DRV_HANDLE hMac);
```

Returns

None

Description

This function closes a client instance of the PIC32 MAC Driver.

Remarks

None

Preconditions

DRV_ETHMAC_PIC32MACOpen() should have been called.

Example

Parameters

Parameters	Description
hMac	valid MAC handle, obtained by a call to DRV_ETHMAC_PIC32MACOpen

Function

void DRV_ETHMAC_PIC32MACClose(DRV_HANDLE hMac)

DRV_ETHMAC_PIC32MACDeinitialize Function

Deinitializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACDeinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function supports teardown of the PIC32 Ethernet MAC (opposite of set up). Used by tcpip_module_manager.

Remarks

This function deinitializes the Ethernet controller, the MAC and the associated PHY. It should be called to be release any resources allocated by the initialization and return the MAC and the PHY to the idle/power down state.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver.

Example

Function

void DRV_ETHMAC_PIC32MACDeinitialize(SYS_MODULE_OBJ object);

DRV_ETHMAC_PIC32MACInitialize Function

Initializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
SYS_MODULE_OBJ DRV_ETHMAC_PIC32MACInitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

- a valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

This function supports the initialization of the PIC32 Ethernet MAC. Used by tcpip_module_manager.

Remarks

This function initializes the Ethernet controller, the MAC and the associated PHY. It should be called to be able to schedule any Ethernet transmit or receive operation.

Preconditions

None

Example

Function

SYS_MODULE_OBJ DRV_ETHMAC_PIC32MACInitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);

DRV_ETHMAC_PIC32MACLinkCheck Function

Checks current link status.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACLinkCheck(DRV_HANDLE hMac);
```

Returns

- true - If the link is up
- false - If the link is not up

Description

This function checks the link status of the associated network interface.

Remarks

The function will automatically perform a MAC reconfiguration if the link went up after being down and the PHY auto negotiation is enabled.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver. DRV_ETHMAC_PIC32MACOpen() should have been called to

obtain a valid handle.

Example

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Function

bool DRV_ETHMAC_PIC32MACLinkCheck(DRV_HANDLE hMac)

DRV_ETHMAC_PIC32MACOpen Function

Opens a client instance of the PIC32 MAC Driver.

Implementation: Dynamic

File

drv_ethmac.h

C

```
DRV_HANDLE DRV_ETHMAC_PIC32MACOpen(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE - handle (pointer) to MAC client
- 0 if call failed

Description

This function opens a client instance of the PIC32 MAC Driver. Used by tcpip_module_manager.

Remarks

The intent parameter is not used in the current implementation and is maintained only for compatibility with the generic driver Open function signature.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called.

Example

Function

DRV_HANDLE DRV_ETHMAC_PIC32MACOpen(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

DRV_ETHMAC_PIC32MACParametersGet Function

MAC parameter get function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_OK if pMacParams updated properly
- a TCPIP_MAC_RES error code if processing failed for some reason

Description

MAC Parameter Get function TCPIP_MAC_RES DRV_ETHMAC_PIC32MACParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);

This is a function that returns the run time parameters of the MAC driver.

Remarks

None.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

DRV_ETHMAC_PIC32MACPowerMode Function

Selects the current power mode for the Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACPowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- true if the call succeeded.
- false if the call failed

Description

This function sets the power mode for the Ethernet MAC.

Remarks

This function is not currently supported by the Ethernet MAC and will always return true.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

bool DRV_ETHMAC_PIC32MACPowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode)

DRV_ETHMAC_PIC32MACProcess Function

MAC periodic processing function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACProcess(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK
- a TCPIP_MAC_RES error code if processing failed for some reason

Description

This is a function that allows for internal processing by the MAC driver. It is meant for processing that cannot be done from within ISR.

Normally this function will be called in response to an TX and/or RX event signaled by the driver. This is specified by the MAC driver at initialization time using TCPIP_MAC_MODULE_CTRL.

Remarks

- The MAC driver may use the DRV_ETHMAC_PIC32MACProcess() for:
- Processing its pending TX queues
- RX buffers replenishing functionality. If the number of packets in the RX queue falls below a specified limit, the MAC driver may use this

function to allocate some extra RX packets. Similarly, if there are too many allocated RX packets, the MAC driver can free some of them.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACProcess(DRV_HANDLE hMac);

DRV_ETHMAC_PIC32MACStatisticsGet Function

Gets the current MAC statistics.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK.
- TCPIP_MAC_RES_OP_ERR error code if function not supported by the driver.

Description

This function will get the current value of the statistic counters maintained by the MAC driver.

Remarks

- The reported values are info only and change dynamically.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics, TCPIP_MAC_TX_STATISTICS* pTxStatistics);

DRV_ETHMAC_PIC32MACStatus Function

Provides the current status of the MAC driver module.

Implementation: Dynamic

File

drv_ethmac.h

C

```
SYS_STATUS DRV_ETHMAC_PIC32MACStatus(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the MAC driver module.

Remarks

None.

Preconditions

The DRV_ETHMAC_PIC32MACInitialize function must have been called before calling this function.

Example

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHMAC_PIC32MACInitialize

Function

SYS_STATUS DRV_ETHMAC_PIC32MACStatus (SYS_MODULE_OBJ object)

DRV_ETHMAC_PIC32MACConfigGet Function

Gets the current MAC driver configuration.

Implementation: Dynamic

File

drv_ethmac.h

C

```
size_t DRV_ETHMAC_PIC32MACConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t*
pConfigSize);
```

Returns

- number of bytes copied into the supplied storage buffer

Description

This function will get the current MAC driver configuration and store it into a supplied buffer.

Remarks

- None

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

size_t DRV_ETHMAC_PIC32MACConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);

DRV_ETHMAC_PIC32MACRegisterStatisticsGet Function

Gets the current MAC hardware statistics registers.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK.
- TCPIP_MAC_RES_OP_ERR error code if function not supported by the driver.

Description

This function will get the current value of the statistic registers of the associated MAC controller.

Remarks

- The reported values are info only and change dynamically.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

DRV_ETHMAC_PIC32MACReinitialize Function

Reinitializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACReinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function supports re-initialization of the PIC32 Ethernet MAC (opposite of set up).

Remarks

This function is not supported yet.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver.

Example

Function

void DRV_ETHMAC_PIC32MACReinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);

b) Receive Functions

DRV_ETHMAC_PIC32MACPacketRx Function

This is the MAC receive function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_PACKET* DRV_ETHMAC_PIC32MACPacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- a valid pointer to an available RX packet
- 0 if no packet pending/available

Description

This function will return a packet if such a pending packet exists.

Additional information about the packet is available by providing the pRes and ppPktStat fields.

Remarks

- Once a pending packet is available in the MAC driver internal RX queues this function will dequeue the packet and hand it over to the MAC driver's client - i.e., the stack - for further processing.
- The flags for a RX packet are updated by the MAC driver:
- TCPIP_MAC_PKT_FLAG_RX will be set
- TCPIP_MAC_PKT_FLAG_UNICAST is set if that packet is a unicast packet
- TCPIP_MAC_PKT_FLAG_BCAST is set if that packet is a broadcast packet
- TCPIP_MAC_PKT_FLAG_MCAST is set if that packet is a multicast packet
- TCPIP_MAC_PKT_FLAG_QUEUED is set
- TCPIP_MAC_PKT_FLAG_SPLIT is set if the packet has multiple data segments
- The MAC driver dequeues and return to the caller just one single packet. That is the packets are not chained.
- The packet buffers are allocated by the Ethernet MAC driver itself, Once the higher level layers in the stack are done with processing the RX packet, they have to call the corresponding packet acknowledgment function that tells the MAC driver that it can resume control of that packet.
- Once the stack modules are done processing the RX packets and the acknowledge function is called the MAC driver will reuse the RX packets.
- The MAC driver may use the DRV_ETHMAC_PIC32MACProcess() for obtaining new RX packets if needed.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_PACKET* DRV_ETHMAC_PIC32MACPacketRx (DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const TCPIP_MAC_PACKET_RX_STAT** ppPktStat);

DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet Function

Sets the current MAC hash table receive filter.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet(DRV_HANDLE hMac, const TCPIP_MAC_ADDR*
DestMACAddr);
```

Returns

- TCPIP_MAC_RES_OK if success
- a TCPIP_MAC_RES error value if failed

Description

This function sets the MAC hash table filtering to allow packets sent to DestMACAddr to be received. It calculates a CRC-32 using polynomial 0x4C11DB7 over the 6 byte MAC address and then, using bits 28:23 of the CRC, will set the appropriate bits in the hash table filter registers (ETHHT0-ETHHT1).

The function will enable/disable the Hash Table receive filter if needed.

Remarks

- Sets the appropriate bit in the ETHHT0/1 registers to allow packets sent to DestMACAddr to be received and enabled the Hash Table receive filter.
- There is no way to individually remove destination MAC addresses from the hash table since it is possible to have a hash collision and therefore multiple MAC addresses relying on the same hash table bit.

- A workaround is to have the stack store each enabled MAC address and to perform the comparison at run time.
- A call to DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet() using a 00-00-00-00-00-00 destination MAC address, which will clear the entire hash table and disable the hash table filter. This will allow the receive of all packets, regardless of their destination

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet(DRV_HANDLE hMac, const TCPIP_MAC_ADDR* DestMACAddr)

c) Transmit Functions

DRV_ETHMAC_PIC32MACPacketTx Function

MAC driver transmit function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACPacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OK if success
- a TCPIP_MAC_RES error value if failed

Description

This is the MAC transmit function. Using this function a packet is submitted to the MAC driver for transmission.

Remarks

- The MAC driver supports internal queuing. A packet is rejected only if it's not properly formatted. Otherwise it will be scheduled for transmission and queued internally if needed.
- Once the packet is scheduled for transmission the MAC driver will set the TCPIP_MAC_PKT_FLAG_QUEUED flag so that the stack is aware that this packet is under processing and cannot be modified.
- Once the packet is transmitted, the TCPIP_MAC_PKT_FLAG_QUEUED will be cleared, the proper packet acknowledgment result (ackRes) will be set and the packet acknowledgment function (ackFunc) will be called.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACPacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);

d) Event Functions

DRV_ETHMAC_PIC32MACEventAcknowledge Function

Acknowledges and re-enables processed events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACEventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT tcpAckEv);
```

Returns

- true if events acknowledged
- false if no events to be acknowledged

Description

This function acknowledges and re-enables processed events. Multiple events can be ORed together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the stack by calling DRV_ETHMAC_PIC32MACEventPendingGet() or have been passed to the stack by the driver using the registered notification handler and have been processed and have to be re-enabled.

Remarks

- All events should be acknowledged, in order to be re-enabled.
- Some events are fatal errors and should not be acknowledged (TCPIP_MAC_EV_RX_BUSERR, TCPIP_MAC_EV_TX_BUSERR). Driver/stack re-initialization is needed under such circumstances.
- Some events are just system/application behavior and they are intended only as simple info (TCPIP_MAC_EV_RX_OVFLOW, TCPIP_MAC_EV_RX_BUFNA, TCPIP_MAC_EV_TX_ABORT, TCPIP_MAC_EV_RX_ACT).
- The TCPIP_MAC_EV_RX_FWMARK and TCPIP_MAC_EV_RX_EWMARK events are part of the normal flow control operation (if auto flow control was enabled). They should be enabled alternatively, if needed.
- The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
DRV_ETHMAC_PIC32MACEventAcknowledge( hMac, stackNewEvents );
```

Function

bool DRV_ETHMAC_PIC32MACEventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT tcpAckEv);

DRV_ETHMAC_PIC32MACEventMaskSet Function

Enables/disables the MAC events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACEventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

always true, operation succeeded.

Description

This is a function that enables or disables the events to be reported to the Ethernet MAC client (TCP/IP stack).

All events that are to be enabled will be added to the notification process. All events that are to be disabled will be removed from the notification process. The stack has to catch the events that are notified and process them. After that the stack should call DRV_ETHMAC_PIC32MACEventAcknowledge() so that the events can be re-enable

The stack should process at least the following transfer events:

- TCPIP_MAC_EV_RX_PKTPEND
- TCPIP_MAC_EV_RX_DONE
- TCPIP_MAC_EV_TX_DONE

Remarks

- The event notification system enables the user of the TCP/IP stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop.
- If the notification events are nil, the interrupt processing will be disabled. Otherwise, the event notification will be enabled and the interrupts relating to the requested events will be enabled.
- Note that once an event has been caught by the stack ISR (and reported if a notification handler is in place) it will be disabled until the DRV_ETHMAC_PIC32MACEventAcknowledge() is called.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
DRV_ETHMAC_PIC32MACEventMaskSet( hMac, TCPIP_MAC_EV_RX_OVFLOW | TCPIP_MAC_EV_RX_BUFNA, true );
```

Function

bool DRV_ETHMAC_PIC32MACEventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);

DRV_ETHMAC_PIC32MACEventPendingGet Function

Returns the currently pending events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_EVENT DRV_ETHMAC_PIC32MACEventPendingGet(DRV_HANDLE hMac);
```

Returns

The currently stack pending events.

Description

This function returns the currently pending Ethernet MAC events. Multiple events will be ORed together as they accumulate. The stack should perform processing whenever a transmission related event (TCPIP_MAC_EV_RX_PKTPEND, TCPIP_MAC_EV_TX_DONE) is present. The other, non critical events, may not be managed by the stack and passed to an user. They will have to be eventually acknowledged if re-enabling is needed.

Remarks

- This is the preferred method to get the current pending MAC events. The stack maintains a proper image of the events from their occurrence to their acknowledgment.
- Even with a notification handler in place it's better to use this function to get the current pending events rather than using the events passed by the notification handler which could be stale.
- The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.
- The returned value is just a momentary value. The pending events can change any time.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EVENT currEvents = DRV_ETHMAC_PIC32MACEventPendingGet( hMac);
```

Function

TCPIP_MAC_EVENT DRV_ETHMAC_PIC32MACEventPendingGet(DRV_HANDLE hMac)

e) Other Functions

DRV_ETHMAC_Tasks_ISR Function

Ethernet MAC driver interrupt function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_Tasks_ISR(SYS_MODULE_OBJ macIndex);
```

Returns

None.

Description

This is the Ethernet MAC driver interrupt service routine. It processes the Ethernet related interrupts and notifies the events to the driver user (the TCP/IP stack).

Remarks

None.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. The TCP/IP stack event notification should be enabled.

Function

void DRV_ETHMAC_Tasks_ISR(SYS_MODULE_OBJ macIndex)

DRV_ETHMAC_PIC32MACTasks Function

Maintains the EThernet MAC driver's state machine.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACTasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine

Remarks

None.

Preconditions

The DRV_ETHMAC_PIC32MACInitialize routine must have been called for the specified MAC driver instance.

Example

Function

void DRV_ETHMAC_PIC32MACTasks(SYS_MODULE_OBJ object)

f) Data Types and Constants

DRV_ETHMAC_INDEX_1 Macro

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_1 1
```

Description

This is macro DRV_ETHMAC_INDEX_1.

DRV_ETHMAC_INDEX_0 Macro

Ethernet driver index definitions.

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_0 0
```

Description

Ethernet Driver Module Index Numbers

These constants provide Ethernet driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the MAC initialization routines to identify the driver instance in use.

DRV_ETHMAC_INDEX_COUNT Macro

Number of valid Ethernet driver indices.

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_COUNT ETH_NUMBER_OF_MODULES
```

Description

Ethernet Driver Module Index Count

This constant identifies number of valid Ethernet driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_ethmac.h	Ethernet MAC device driver interface file
drv_ethmac_config.h	Ethernet MAC driver configuration definitions template.

Description

This section lists the source and header files used by the Ethernet MAC Driver Library.

drv_ethmac.h

Ethernet MAC device driver interface file

Functions

	Name	Description
	DRV_ETHMAC_PIC32MACClose	Closes a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACConfigGet	Gets the current MAC driver configuration. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACDeinitialize	Deinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventAcknowledge	Acknowledges and re-enables processed events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventMaskSet	Enables/disables the MAC events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventPendingGet	Returns the currently pending events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACInitialize	Initializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACLinkCheck	Checks current link status. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACOpen	Opens a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPacketRx	This is the MAC receive function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPacketTx	MAC driver transmit function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACParametersGet	MAC parameter get function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPowerMode	Selects the current power mode for the Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACProcess	MAC periodic processing function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRegisterStatisticsGet	Gets the current MAC hardware statistics registers. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACReinitialize	Reinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet	Sets the current MAC hash table receive filter. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatisticsGet	Gets the current MAC statistics. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatus	Provides the current status of the MAC driver module. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACTasks	Maintains the EThernet MAC driver's state machine. **Implementation:** Dynamic
	DRV_ETHMAC_Tasks_ISR	Ethernet MAC driver interrupt function. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_ETHMAC_INDEX_0	Ethernet driver index definitions.
	DRV_ETHMAC_INDEX_1	This is macro DRV_ETHMAC_INDEX_1.
	DRV_ETHMAC_INDEX_COUNT	Number of valid Ethernet driver indices.

Description

Ethernet MAC Device Driver Interface

The Ethernet MAC device driver provides a simple interface to manage the Ethernet peripheral. This file defines the interface definitions and prototypes for the Ethernet MAC driver.

File Name

drv_ethmac.h

Company

Microchip Technology Inc.

drv_ethmac_config.h

Ethernet MAC driver configuration definitions template.

Macros

	Name	Description
	DRV_ETHMAC_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHMAC_INDEX	Ethernet MAC static index selection.
	DRV_ETHMAC_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHMAC_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ETHMAC_INTERRUPT_SOURCE	Defines an override of the interrupt source in case of static driver.
	DRV_ETHMAC_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHMAC_POWER_STATE	Defines an override of the power state of the Ethernet MAC driver.

Description

ETHMAC Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

File Name

drv_ethmac_config.h

Company

Microchip Technology Inc.

Ethernet PHY Driver Library

This section describes the Ethernet PHY Driver Library.

Introduction

This library provides a low-level abstraction of the Ethernet PHY Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

This library provides a software abstraction for configuring external Ethernet PHY devices for use with the on-chip PIC32 Ethernet Controller.

Using the Library

The user of this driver is the MPLAB Harmony TCP/IP Stack through its Ethernet MAC driver. This Ethernet PHY driver is not intended as a system wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the PHY driver required by the Ethernet MAC.

This topic describes the basic architecture and functionality of the Ethernet PHY driver and is meant for advanced users or TCP/IP Stack driver developers.

Interface Header File: `drv_ethphy.h`

The interface to the Ethernet PHY library is defined in the `drv_ethphy.h` header file, which is included by the MPLAB Harmony TCP/IP stack.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

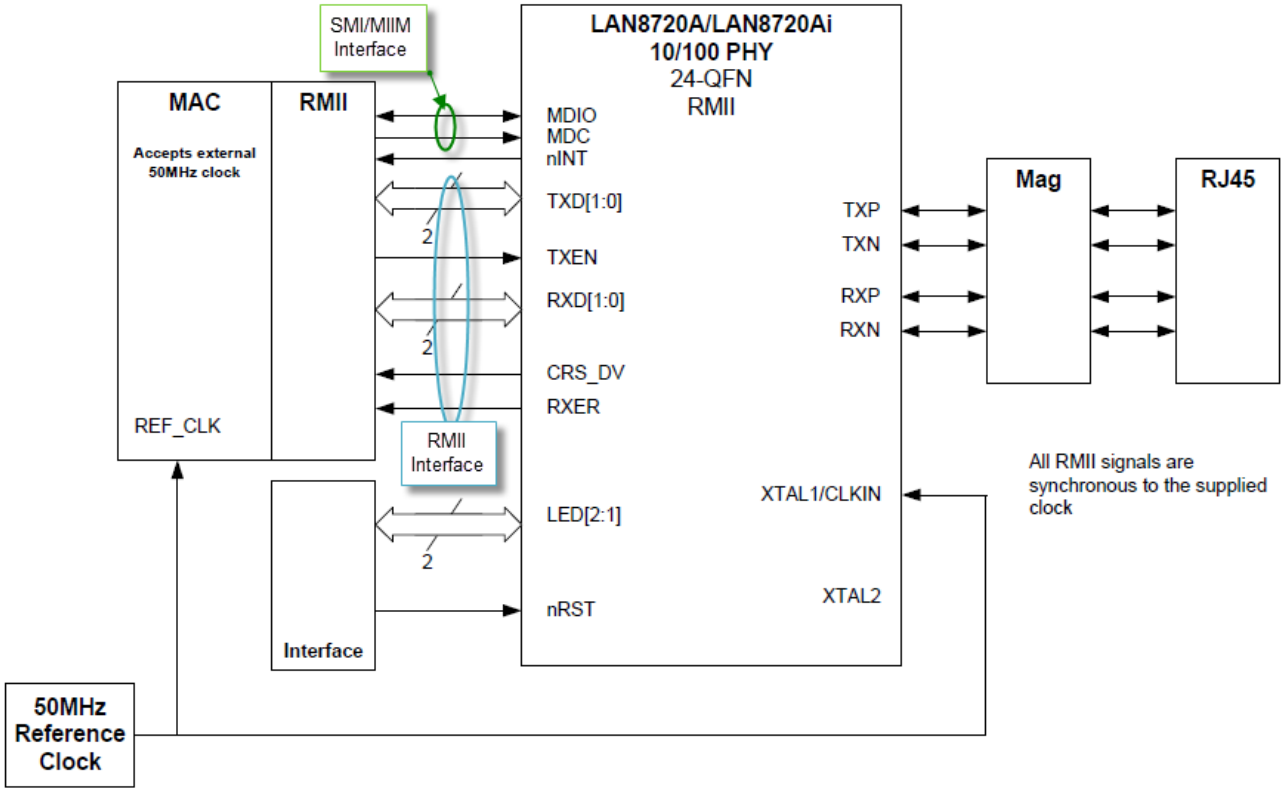
Abstraction Model

This library provides a low-level abstraction of the Ethernet PHY Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

To understand how this library works you must first understand how an external Ethernet PHY interfaces with the Ethernet Controller. As shown in Figure 1, the PHY has two interfaces, one for managing the PHY, known as the Serial Management Interface (SMI), for configuring the device and a second, known as the Reduced Media Independent Interface (RMII), for transmit and receive data.

Figure 1: Typical External PHY Interface



The block diagram also shows an interrupt signal (nINT) going to a external interrupt pin on the host device and signals going to on-board LEDs to show link state and link activity.

The SMI interface is also known as the MII Management (MIIM) interface. This control interface is standardized for all PHYs by Clause 22 of the 802.3 standard. It provides up to 32 16-bit registers on the PHY. The following table provides a summary of all 32 registers. Consult the data sheet for the PHY device for the specific bit fields in each register.

Register Address	Register Name	Register Type
0	Control	Basic
1	Status	Basic
2, 3	PHY Identifier	Extended
4	Auto-Negotiation Advertisement	Extended
5	Auto-Negotiation Link Partner Base Page Ability	Extended
6	Auto-Negotiation Expansion	Extended
7	Auto-Negotiation Next Page Transmit	Extended

8	Auto-Negotiation Link Partner Received Next Page	Extended
9	MASTER-SLAVE Control Register	Extended
10	MASTER-SLAVE Status Register	Extended
11-14	Reserved	Extended
15	Extended Status	Reserved
16-31	Vendor Specific	Extended

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet PHY Driver Library

Library Interface Section	Description
System Level Functions	Routines that integrate the driver into the MPLAB Harmony framework.
Client Level Functions	Open, Close, Link Status, Auto Negotiation.
SMI/MIIM Functions	SMI/MIIM Management Interface.
External PHY Support Functions	Provides the API for PHY support routines that the driver will call when setting up the PHY. The driver library provides support for four PHYs.
Other Functions	Functions that provide software version information.
Data Types and Constants	C language typedefs and enums used by this library.

Configuring the Library

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHPHY_NEG_DONE_TMO	Value of the PHY negotiation complete time out as per IEEE 802.3 spec.
	DRV_ETHPHY_NEG_INIT_TMO	Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.
	DRV_ETHPHY_RESET_CLR_TMO	Value of the PHY Reset self clear time out as per IEEE 802.3 spec.

Description

The configuration of the Ethernet PHY Driver Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Ethernet PHY Driver Library. Based on the selections made, the Ethernet PHY Driver Library may support the selected features.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ETHPHY_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_CLIENTS_NUMBER 1
```

Description

Ethernet PHY Maximum Number of Clients This definition select the maximum number of clients that the Ethernet PHY driver can support at run time. Not defining it means using a single client.

Remarks

The MAC driver is the client of the PHY driver. Multiple clients may be needed when access to MIIM bus (for PHY vendor specific functionality) is needed through the PHY driver.

However MIIM operations are not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. In this case the number of clients should be 1 and the DRV_MIIM should be used for accessing the MIIM bus.

DRV_ETHPHY_INDEX Macro

Ethernet PHY static index selection.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_INDEX DRV_ETHPHY_INDEX_1
```

Description

Ethernet PHY Static Index Selection

This definition selects the Ethernet PHY static index for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_ETHPHY_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_INSTANCES_NUMBER 1
```

Description

Ethernet PHY hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

DRV_ETHPHY_PERIPHERAL_ID Macro

Defines an override of the peripheral ID.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_PERIPHERAL_ID ETHPHY_ID_1
```

Description

Ethernet PHY Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHPHY_ID_0

DRV_ETHPHY_NEG_DONE_TMO Macro

Value of the PHY negotiation complete time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_NEG_DONE_TMO (2000)
```

Description

Ethernet PHY Negotiation Complete time out

This definition sets the time out of the PHY negotiation complete, in ms.

Remarks

See IEEE 802.3 Clause 28 Table 28-9 autoneg_wait_timer value (max 1s).

DRV_ETHPHY_NEG_INIT_TMO Macro

Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_NEG_INIT_TMO (1)
```

Description

Ethernet PHY Negotiation Initiation time out

This definition sets the time out of the PHY negotiation initiation, in ms.

Remarks

None.

DRV_ETHPHY_RESET_CLR_TMO Macro

Value of the PHY Reset self clear time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_RESET_CLR_TMO (500)
```

Description

Ethernet PHY Reset self clear time out

This definition sets the time out of the PHY Reset self clear, in ms.

Remarks

See IEEE 802.3 Clause 22 Table 22-7 and paragraph "22.2.4.1.1 Reset" (max 0.5s)

Building the Library

This section lists the files that are available in the Ethernet PHY Driver Library.

Description

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is

`<install-dir>/framework/driver/ethphy`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ethphy.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_extphy.c`	Basic PHY driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_extphy_smsc8700.c`	SMSC 8700 PHY implementation file.
`/src/dynamic/drv_extphy_smsc8720.c`	SMSC 8720 PHY implementation file.
`/src/dynamic/drv_extphy_smsc8720.c`	SMSC 8740 PHY implementation file.
`/src/dynamic/drv_extphy_ip101gr.c`	IP101GR PHY implementation file.
`/src/dynamic/drv_extphy_dp83640.c`	National DP83640 PHY implementation file.
`/src/dynamic/drv_extphy_dp83848.c`	National DP83848 PHY implementation file.

Module Dependencies

The Ethernet MAC Driver Library depends on the following modules:

- Ethernet MAC Driver Library
- Clock System Service Library
- Ports System Service Library
- Timer System Service Library
- Ethernet Peripheral Library

Library Interface

a) System Level Functions

	Name	Description
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags. **Implementation:** Dynamic
	DRV_ETHPHY_Setup	Initializes Ethernet PHY configuration and set up procedure. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationAbort	Aborts a current client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationResult	Gets the result of a client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic

c) SMI/MIIM Functions

	Name	Description
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStop	Stops the scan of a previously requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStart	Starts the scan of a requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIRead	Initiates a SMI/MIIM read transaction. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result. **Implementation:** Dynamic
	DRV_ETHPHY_SMIStatus	Returns the current status of the SMI/MIIM interface. **Implementation:** Dynamic
	DRV_ETHPHY_SMIWrite	Initiates a SMI/MIIM write transaction. **Implementation:** Dynamic

d) Vendor Functions

	Name	Description
	DRV_ETHPHY_VendorDataGet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataSet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadResultGet	Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadStart	Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIWriteStart	Starts a vendor SMI write transfer. **Implementation:** Dynamic

e) Other Functions

	Name	Description
	DRV_ETHPHY_LinkStatusGet	Returns the current link status. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationResultGet	Returns the result of a completed negotiation. **Implementation:** Dynamic

	DRV_ETHPHY_PhyAddressGet	Returns the PHY address. **Implementation:** Dynamic
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.
	DRV_ETHPHY_NEGOTIATION_RESULT	Contains all the data necessary to get the Ethernet PHY negotiation result
	DRV_ETHPHY_SETUP	Contains all the data necessary to set up the Ethernet PHY device.
	DRV_ETHPHY_VENDOR_MDIX_CONFIGURE	Pointer to function that configures the MDIX mode for the Ethernet PHY.
	DRV_ETHPHY_VENDOR_MII_CONFIGURE	Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.
	DRV_ETHPHY_VENDOR_SMI_CLOCK_GET	Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.
	DRV_ETHPHY_LINK_STATUS	Defines the possible status flags of PHY Ethernet link.
	DRV_ETHPHY_CONFIG_FLAGS	Defines configuration options for the Ethernet PHY.
	DRV_ETHPHY_OBJECT	Identifies the interface of a Ethernet PHY vendor driver.
	DRV_ETHPHY_VENDOR_WOL_CONFIGURE	Pointer to a function to configure the PHY WOL functionality
	DRV_ETHPHY_OBJECT_BASE_TYPE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_OBJECT_BASE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_RESET_FUNCTION	Pointer to a function to perform an additional PHY reset
	DRV_ETHPHY_RESULT	Defines the possible results of Ethernet operations that can succeed or fail
	DRV_ETHPHY_USE_DRV_MIIM	Defines the way the PHY driver accesses the MIIM bus to communicate with the PHY.
	DRV_ETHPHY_INTERFACE_INDEX	Defines the index type for a PHY interface.
	DRV_ETHPHY_INTERFACE_TYPE	Defines the type of interface a PHY supports.

Description

This section describes the Application Programming Interface (API) functions of the Ethernet PHY Driver Library.

Refer to each section for a detailed description.

a) System Level Functions

DRV_ETHPHY_Initialize Function

Initializes the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
SYS_MODULE_OBJ DRV_ETHPHY_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- a valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

This function initializes the Ethernet PHY driver, making it ready for clients to open and use it.

Remarks

- This function must be called before any other Ethernet PHY routine is called.

- This function should only be called once during system initialization unless DRV_ETHPHY_Deinitialize is called to deinitialize the driver instance.
- The returned object must be passed as argument to DRV_ETHPHY_Reinitialize, DRV_ETHPHY_Deinitialize, DRV_ETHPHY_Tasks and DRV_ETHPHY_Status routines.

Preconditions

None.

Example

```
DRV_ETHPHY_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_0;

// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_2;
init.pPhyObject  = &DRV_ETHPHY_OBJECT_SMSC_LAN8720;

// Do something

objectHandle = DRV_ETHPHY_Initialize(DRV_ETHPHY_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Function

SYS_MODULE_OBJ DRV_ETHPHY_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_ETHPHY_Deinitialize Function

Deinitializes the specified instance of the Ethernet PHY driver module.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the Ethernet PHY driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

- Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_ETHPHY_Initialize
SYS_STATUS          status;

DRV_ETHPHY_Deinitialize(object);

status = DRV_ETHPHY_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
```

```
    // when the driver is deinitialized.
}
```

Function

void DRV_ETHPHY_Deinitialize (SYS_MODULE_OBJ object)

DRV_ETHPHY_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Remarks

- This function can be called multiple times to reinitialize the module.
- This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_ETHPHY_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_2;
init.pPhyObject  = &DRV_ETHPHY_OBJECT_SMSC_LAN8720;

DRV_ETHPHY_Reinitialize(objectHandle, (SYS_MODULE_INIT*)&init);

phyStatus = DRV_ETHPHY_Status(objectHandle);
if (SYS_STATUS_BUSY == phyStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= phyStatus)
{
    // Handle error
}
```

Function

void DRV_ETHPHY_Reinitialize(SYS_MODULE_OBJ object,

const SYS_MODULE_INIT * const init)

DRV_ETHPHY_Status Function

Provides the current status of the Ethernet PHY driver module.

Implementation: Dynamic

File

drv_ethphy.h

C

```
SYS_STATUS DRV_ETHPHY_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the Ethernet PHY driver module.

Remarks

- Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.
- SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another
- SYS_STATUS_ERROR - Indicates that the driver is in an error state
- Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized
- The this operation can be used to determine when any of the driver's module level operations has completed.
- If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.
- The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.
- This function will NEVER block waiting for hardware.
- If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_ETHPHY_Initialize
SYS_STATUS          status;

status = DRV_ETHPHY_Status(object);
if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHPHY_Initialize

Function

SYS_STATUS DRV_ETHPHY_Status (SYS_MODULE_OBJ object)

DRV_ETHPHY_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Remarks

- This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)
- This function will never block or access any resources that may cause it to block.

Preconditions

The DRV_ETHPHY_Initialize routine must have been called for the specified Ethernet PHY driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_ETHPHY_Initialize

while (true)
{
    DRV_ETHPHY_Tasks (object);

    // Do other tasks
}
```

Function

void DRV_ETHPHY_Tasks(SYS_MODULE_OBJ object)

DRV_ETHPHY_HWConfigFlagsGet Function

Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_HWConfigFlagsGet(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags);
```

Returns

DRV_ETHPHY_RES_OK - if the configuration flags successfully stored at pFlags DRV_ETHPHY_RESULT error code otherwise

Description

This function returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags from the Device Configuration Fuse bits.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
pFlags	address to store the hardware configuration

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_HWConfigFlagsGet(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags)

DRV_ETHPHY_Setup Function

Initializes Ethernet PHY configuration and set up procedure.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_Setup(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS*
pSetupFlags);
```

Returns

- DRV_ETHPHY_RES_PENDING operation has been scheduled successfully
- an DRV_ETHPHY_RESULT error code if the set up procedure failed.

Description

This function initializes the Ethernet PHY communication. It tries to detect the external Ethernet PHY, to read the capabilities and find a match with the requested features. Then, it programs the Ethernet PHY accordingly.

Remarks

PHY configuration may be a lengthy operation due to active negotiation that the PHY has to perform with the link party. The DRV_ETHPHY_ClientStatus will repeatedly return DRV_ETHPHY_CLIENT_STATUS_BUSY until the set up procedure is complete (unless an error detected at which an error code will be returned immediately).

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_Setup(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS* pSetupFlags)

b) Client Level Functions

DRV_ETHPHY_ClientStatus Function

Gets the current client-specific status the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_CLIENT_STATUS DRV_ETHPHY_ClientStatus(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the Ethernet PHY driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

This function has to be used to check that a driver operation has completed. It will return DRV_ETHPHY_CLIENT_STATUS_BUSY when an operation is in progress. It will return DRV_ETHPHY_CLIENT_STATUS_READY when the operation has completed.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE phyHandle;  // Returned from DRV_ETHPHY_Open
```

```
DRV_ETHPHY_CLIENT_STATUS phyClientStatus;

phyClientStatus = DRV_ETHPHY_ClientStatus(phyHandle);
if(DRV_ETHPHY_CLIENT_STATUS_ERROR >= phyClientStatus)
{
    // Handle the error
}
```

Function

DRV_ETHPHY_CLIENT_STATUS DRV_ETHPHY_ClientStatus(DRV_HANDLE handle)

DRV_ETHPHY_Close Function

Closes an opened instance of the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the Ethernet PHY driver, invalidating the handle.

Remarks

- After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_ETHPHY_Open before the caller may use the driver again.
- Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_ETHPHY_Initialize routine must have been called for the specified Ethernet PHY driver instance.

DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_ETHPHY_Open

DRV_ETHPHY_Close(handle);
```

Function

void DRV_ETHPHY_Close(DRV_HANDLE handle)

DRV_ETHPHY_Open Function

Opens the specified Ethernet PHY driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_HANDLE DRV_ETHPHY_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- valid open-instance handle if successful (a number identifying both the caller and the module instance).
- DRV_HANDLE_INVALID if an error occurs

Description

This function opens the specified Ethernet PHY driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_ETHPHY_Close routine is called.

This function will NEVER block waiting for hardware.

The intent parameter is not used. The PHY driver implements a non-blocking behavior.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_ETHPHY_Open(DRV_ETHPHY_INDEX_0, 0);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Function

DRV_HANDLE DRV_ETHPHY_Open(const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_ETHPHY_Reset Function

Immediately resets the Ethernet PHY.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_Reset(DRV_HANDLE handle, bool waitComplete);
```

Returns

- DRV_ETHPHY_RES_PENDING for ongoing, in progress operation
- DRV_ETHPHY_RES_OPERATION_ERR - invalid parameter or operation in the current context

Description

This function immediately resets the Ethernet PHY, optionally waiting for a reset to complete.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_DTCT_ERR if the PHY failed to respond

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_Reset(DRV_HANDLE handle, bool waitComplete)

DRV_ETHPHY_ClientOperationAbort Function

Aborts a current client operation initiated by the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationAbort(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RESULT value describing the current operation result: DRV_ETHPHY_RES_OK for success; operation has been aborted an DRV_ETHPHY_RESULT error code if the operation failed.

Description

Aborts a current client operation initiated by the Ethernet PHY driver.

Remarks

None

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.
- A driver operation was started

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationAbort(DRV_HANDLE handle)

DRV_ETHPHY_ClientOperationResult Function

Gets the result of a client operation initiated by the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationResult(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RESULT value describing the current operation result: DRV_ETHPHY_RES_OK for success; operation has been completed successfully DRV_ETHPHY_RES_PENDING operation is in progress an DRV_ETHPHY_RESULT error code if the operation failed.

Description

Returns the result of a client operation initiated by the Ethernet PHY driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

This function returns the result of the last driver operation. It will return DRV_ETHPHY_RES_PENDING if an operation is still in progress. Otherwise a DRV_ETHPHY_RESULT describing the operation outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.
- A driver operation was started and completed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationResult(DRV_HANDLE handle)

c) SMI/MIIM Functions

DRV_ETHPHY_SMIScanStatusGet Function

Gets the status of the SMI/MIIM scan data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStatusGet(DRV_HANDLE handle);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - scan data is available

DRV_ETHPHY_RES_PENDING - scan data is not yet available

< 0 - an error has occurred and the operation could not be completed

Description

This function gets the status of the SMI/MIIM scan data.

Remarks

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart() has been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStatusGet(DRV_HANDLE handle)

DRV_ETHPHY_SMIScanStop Function

Stops the scan of a previously requested SMI/MIIM register.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStop(DRV_HANDLE handle);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - the scan transaction has been stopped successfully < 0 - an error has occurred and the operation could not be completed

Description

This function stops the current scan of a SMI/MIIM register.

Remarks

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart was called to start a scan

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStop(DRV_HANDLE handle)

DRV_ETHPHY_SMIClockSet Function

Sets the SMI/MIIM interface clock.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIClockSet(DRV_HANDLE handle, uint32_t hostClock, uint32_t maxSMIClock);
```

Returns

DRV_ETHPHY_RES_HANDLE_ERR - passed in handle was invalid

DRV_ETHPHY_RES_OK - operation successful

< 0 - an error has occurred and the operation could not be completed

Description

This function sets SMI/MIIM interface clock base on host clock and maximum supported SMI/MIIM interface clock speed.

Remarks

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIClockSet(DRV_HANDLE handle,

uint32_t hostClock,

uint32_t maxSMIClock)

DRV_ETHPHY_SMIScanStart Function

Starts the scan of a requested SMI/MIIM register.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStart(DRV_HANDLE handle, unsigned int rIx);
```

Returns

DRV_ETHPHY_RES_PENDING - the scan transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function starts the scan of a requested SMI/MIIM register.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

However, the client status will always be DRV_ETHPHY_CLIENT_STATUS_BUSY and the client result will always show DRV_ETHPHY_RES_PENDING for as long as the scan is active. Use DRV_ETHPHY_SMIScanStop() to stop a scan in progress. Use DRV_ETHPHY_SMIScanStatusGet() to check is there is scan data available. Use DRV_ETHPHY_SMIScanDataGet() to retrieve the scan data.

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStart(DRV_HANDLE handle,
unsigned int rIx)

DRV_ETHPHY_SMIRead Function

Initiates a SMI/MIIM read transaction.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIRead(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int phyAdd);
```

Returns

DRV_ETHPHY_RES_PENDING - the transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function initiates a SMI/MIIM read transaction for a given PHY register.

Remarks

In most situations the PHY address to be used for this function should be the one returned by DRV_ETHPHY_PhyAddressGet(). However this function allows using a different PHY address for advanced operation.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIRead(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int phyAdd)

DRV_ETHPHY_SMIScanDataGet Function

Gets the latest SMI/MIIM scan data result.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanDataGet(DRV_HANDLE handle, uint16_t* pScanRes);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - scan data is available and stored at pScanRes DRV_ETHPHY_RES_PENDING - scan data is not yet available

< 0 - an error has occurred and the operation could not be completed

Description

This function gets the latest SMI/MIIM scan data result.

Remarks

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart() has been called
- Data is available if DRV_ETHPHY_SMIScanStatusGet() previously returned DRV_ETHPHY_RES_OK

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanDataGet(DRV_HANDLE handle, uint16_t* pScanRes)

DRV_ETHPHY_SMIStatus Function

Returns the current status of the SMI/MIIM interface.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIStatus(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RES_BUSY - if the SMI/MIIM interface is busy
- DRV_ETHPHY_RES_OK - if the SMI/MIIM is not busy

< 0 - an error has occurred and the operation could not be completed

Description

This function checks if the SMI/MIIM interface is busy with a transaction.

Remarks

This function is info only and returns the momentary status of the SMI bus. Even if the bus is free there is no guarantee it will be free later on especially if the driver is on going some operation.

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIStatus(DRV_HANDLE handle)

DRV_ETHPHY_SMIWrite Function

Initiates a SMI/MIIM write transaction.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIWrite(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int phyAdd, bool waitComplete);
```

Returns

DRV_ETHPHY_RES_OK - the write transaction has been scheduled/completed successfully DRV_ETHPHY_RES_PENDING - the transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function initiates a SMI/MIIM write transaction for a given PHY register.

Remarks

In most situations the PHY address to be used for this function should be the one returned by DRV_ETHPHY_PhyAddressGet(). However this function allows using a different PHY address for advanced operation.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

This operation is not supported when the PHY driver uses the MIIM driver for MIIM bus accesses. Use the DRV_MIIM for accessing the MIIM bus.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIWrite(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int phyAdd, bool waitComplete)

d) Vendor Functions

DRV_ETHPHY_VendorDataGet Function

Returns the current value of the vendor data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataGet(DRV_HANDLE handle, uint32_t* pVendorData);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor data is stored at the pVendorData address

DRV_ETHPHY_RES_HANDLE_ERR - handle error

Description

This function returns the current value of the vendor data. Each DRV_ETHPHY client object maintains data that could be used for vendor specific operations. This routine allows retrieving of the vendor specific data.

Remarks

The PHY driver will clear the vendor specific data before any call to a vendor specific routine. Otherwise the PHY driver functions do not touch this value.

The DRV_ETHPHY_VendorDataSet can be used for writing data into this field.

Currently only a 32 bit value is supported.

The function is intended for implementing vendor specific functions, like DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure, that need a way of maintaining their own data and state machine.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataGet(DRV_HANDLE handle, uint32_t* pVendorData)

DRV_ETHPHY_VendorDataSet Function

Returns the current value of the vendor data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataSet(DRV_HANDLE handle, uint32_t vendorData);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor data is stored in the client object

DRV_ETHPHY_RES_HANDLE_ERR - handle error

Description

This function returns the current value of the vendor data. Each DRV_ETHPHY client object maintains data that could be used for vendor specific operations. This routine allows retrieving of the vendor specific data.

Remarks

The PHY driver will clear the vendor specific data before any call to a vendor specific routine. Otherwise the PHY driver functions do not touch this value.

The DRV_ETHPHY_VendorDataGet can be used for reading data into this field.

Currently only a 32 bit value is supported.

The function is intended for implementing vendor specific functions, like DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure, that need a way of maintaining their own data and state machine.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataSet(DRV_HANDLE handle, uint32_t vendorData)

DRV_ETHPHY_VendorSMIReadResultGet Function

Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadResultGet(DRV_HANDLE handle, uint16_t* pSmiRes);
```

Returns

DRV_ETHPHY_RES_OK - transaction complete and result deposited at pSmiRes.

DRV_ETHPHY_RES_PENDING - if the vendor transaction is still ongoing The call needs to be retried.

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will return the data of a SMI read transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed
- DRV_ETHPHY_VendorSMIReadStart should have been called to initiate a transfer

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadResultGet(DRV_HANDLE handle, uint16_t* pSmiRes)

DRV_ETHPHY_VendorSMIReadStart Function

Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadStart(DRV_HANDLE handle, uint16_t rIx, int phyAddress);
```

Returns

DRV_ETHPHY_RES_OK - the vendor transaction is started DRV_ETHPHY_VendorSMIReadResultGet() needs to be called for the transaction to complete and to retrieve the result

DRV_ETHPHY_RES_PENDING - the SMI bus is busy and the call needs to be retried

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will start a SMI read transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadStart(DRV_HANDLE handle, uint16_t rIx, int phyAddress)

DRV_ETHPHY_VendorSMIWriteStart Function

Starts a vendor SMI write transfer.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIWriteStart(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int phyAddress);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor SMI write transfer is started

DRV_ETHPHY_RES_PENDING - the SMI bus was busy and the call needs to be retried

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will start a SMI write transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIWriteStart(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int phyAddress)

e) Other Functions

DRV_ETHPHY_LinkStatusGet Function

Returns the current link status.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_LinkStatusGet(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex, DRV_ETHPHY_LINK_STATUS* pLinkStat, bool refresh);
```

Returns

- DRV_ETHPHY_RES_PENDING for ongoing, in progress operation
- an DRV_ETHPHY_RESULT error code if the link status get procedure failed.

Description

This function returns the current link status.

Remarks

This function reads the Ethernet PHY to get current link status. If refresh is specified then, if the link is down a second read will be performed to return the current link status.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_LinkStatusGet(DRV_HANDLE handle, DRV_ETHPHY_LINK_STATUS* pLinkStat, bool refresh)

DRV_ETHPHY_NegotiationIsComplete Function

Returns the results of a previously initiated Ethernet PHY negotiation.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationIsComplete(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex,
bool waitComplete);
```

Returns

- DRV_ETHPHY_RES_PENDING operation is ongoing
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function returns the results of a previously initiated Ethernet PHY negotiation.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but negotiation has failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_NEGOTIATION_INACTIVE if no negotiation in progress
- DRV_ETHPHY_RES_NEGOTIATION_NOT_STARTED if negotiation not yet started yet (means time out if waitComplete was requested)
- DRV_ETHPHY_RES_NEGOTIATION_ACTIVE if negotiation ongoing

(means time out if waitComplete was requested).

See also DRV_ETHPHY_NegotiationResultGet.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_RestartNegotiation should have been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationIsComplete(DRV_HANDLE handle, bool waitComplete)

DRV_ETHPHY_NegotiationResultGet Function

Returns the result of a completed negotiation.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationResultGet(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex,
DRV_ETHPHY_NEGOTIATION_RESULT* pNegResult);
```

Returns

- DRV_ETHPHY_RES_PENDING operation is ongoing
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function returns the PHY negotiation data gathered after a completed negotiation.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but negotiation has failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_NEGOTIATION_INACTIVE if no negotiation in progress
- DRV_ETHPHY_RES_NEGOTIATION_NOT_STARTED if negotiation not yet started yet (means time out if waitComplete was requested)
- DRV_ETHPHY_RES_NEGOTIATION_ACTIVE if negotiation ongoing

The returned value for the negotiation flags is valid only if the negotiation was completed successfully.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_RestartNegotiation, and DRV_ETHPHY_NegotiationIsComplete should have been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationResultGet(DRV_HANDLE handle, DRV_ETHPHY_NEGOTIATION_RESULT* pNegResult)

DRV_ETHPHY_PhyAddressGet Function

Returns the PHY address.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_PhyAddressGet(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex, int* pPhyAddress);
```

Returns

DRV_ETHPHY_RES_OK - operation successful and the PHY address stored at

DRV_ETHPHY_RES_HANDLE_ERR - passed in handle was invalid pPhyAddress

Description

This function returns the current PHY address as set by the DRV_ETHPHY_Setup procedure.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_PhyAddressGet(DRV_HANDLE handle, int* pPhyAddress);

DRV_ETHPHY_RestartNegotiation Function

Restarts auto-negotiation of the Ethernet PHY link.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_RestartNegotiation(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex);
```

Returns

- DRV_ETHPHY_RES_PENDING operation has been scheduled successfully
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function restarts auto-negotiation of the Ethernet PHY link.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_RestartNegotiation(DRV_HANDLE handle)

f) Data Types and Constants

DRV_ETHPHY_CLIENT_STATUS Enumeration

Identifies the client-specific status of the Ethernet PHY driver.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_CLIENT_STATUS_ERROR,
  DRV_ETHPHY_CLIENT_STATUS_CLOSED,
  DRV_ETHPHY_CLIENT_STATUS_BUSY,
  DRV_ETHPHY_CLIENT_STATUS_READY
} DRV_ETHPHY_CLIENT_STATUS;
```

Members

Members	Description
DRV_ETHPHY_CLIENT_STATUS_ERROR	Unspecified error condition
DRV_ETHPHY_CLIENT_STATUS_CLOSED	Client is not open
DRV_ETHPHY_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_ETHPHY_CLIENT_STATUS_READY	Up and running, no operations running

Description

Ethernet PHY Driver Client Status

This enumeration identifies the client-specific status of the Ethernet PHY driver.

Remarks

None.

DRV_ETHPHY_INIT Structure

Contains all the data necessary to initialize the Ethernet PHY device.

File

drv_ethphy.h

C

```
struct DRV_ETHPHY_INIT {
  SYS_MODULE_INIT moduleInit;
  uintptr_t ethphyId;
  uint16_t phyAddress;
  DRV_ETHPHY_CONFIG_FLAGS phyFlags;
  const DRV_ETHPHY_OBJECT* pPhyObject;
  DRV_ETHPHY_RESET_FUNCTION resetFunction;
  const struct DRV_MIIM_OBJECT_BASE* pMiimObject;
  const struct DRV_MIIM_INIT* pMiimInit;
  SYS_MODULE_INDEX miimIndex;
};
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
uintptr_t ethphyId;	Identifies peripheral (PLIB-level) ID
uint16_t phyAddress;	PHY address, as configured on the board. All PHYs respond to address 0
DRV_ETHPHY_CONFIG_FLAGS phyFlags;	PHY configuration
const DRV_ETHPHY_OBJECT* pPhyObject;	Non-volatile pointer to the PHY object providing vendor functions for this PHY
DRV_ETHPHY_RESET_FUNCTION resetFunction;	Function to be called when the PHY is reset/initialized. Could be NULL if no special reset functionality needed - default
const struct DRV_MIIM_OBJECT_BASE* pMiimObject;	Non-volatile pointer to the DRV_MIIM object providing MIIM access for this PHY Could be NULL if the MIIM driver is not used
const struct DRV_MIIM_INIT* pMiimInit;	Non-volatile pointer to the DRV_MIIM initialization data Could be NULL if the MIIM driver is not used
SYS_MODULE_INDEX miimIndex;	MIIM module index to be used Not needed if the MIIM driver is not used

Description

Ethernet PHY Device Driver Initialization Data

This data structure contains all the data necessary to initialize the Ethernet PHY device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_ETHPHY_Initialize routine.

DRV_ETHPHY_NEGOTIATION_RESULT Structure

Contains all the data necessary to get the Ethernet PHY negotiation result

File

drv_ethphy.h

C

```
typedef struct {
  DRV_ETHPHY_LINK_STATUS linkStatus;
  TCPIP_ETH_OPEN_FLAGS linkFlags;
  TCPIP_ETH_PAUSE_TYPE pauseType;
} DRV_ETHPHY_NEGOTIATION_RESULT;
```

Members

Members	Description
DRV_ETHPHY_LINK_STATUS linkStatus;	link status after a completed negotiation

TCPIP_ETH_OPEN_FLAGS linkFlags;	the negotiation result flags
TCPIP_ETH_PAUSE_TYPE pauseType;	pause type supported by the link partner

Description

Ethernet PHY Device Driver Negotiation result Data

Contains all the data necessary to get the Ethernet PHY negotiation result

Remarks

A pointer to a structure of this format must be passed into the DRV_ETHPHY_NegotiationResultGet routine.

DRV_ETHPHY_SETUP Structure

Contains all the data necessary to set up the Ethernet PHY device.

File

drv_ethphy.h

C

```
typedef struct {
  int phyAddress;
  TCPIP_ETH_OPEN_FLAGS openFlags;
  DRV_ETHPHY_CONFIG_FLAGS configFlags;
  TCPIP_ETH_PAUSE_TYPE macPauseType;
  DRV_ETHPHY_RESET_FUNCTION resetFunction;
} DRV_ETHPHY_SETUP;
```

Members

Members	Description
int phyAddress;	the address the PHY is configured for
TCPIP_ETH_OPEN_FLAGS openFlags;	the capability flags: FD/HD, 100/100Mbps, etc.
DRV_ETHPHY_CONFIG_FLAGS configFlags;	configuration flags: MII/RMII, I/O setup
TCPIP_ETH_PAUSE_TYPE macPauseType;	MAC requested pause type
DRV_ETHPHY_RESET_FUNCTION resetFunction;	If ! NULL, function to be called when the PHY is reset/initialized

Description

Ethernet PHY Device Driver Set up Data

This data structure contains all the data necessary to configure the Ethernet PHY device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_ETHPHY_Setup routine.

DRV_ETHPHY_VENDOR_MDIX_CONFIGURE Type

Pointer to function that configures the MDIX mode for the Ethernet PHY.

File

drv_ethphy.h

C

```
typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MDIX_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle, TCPIP_ETH_OPEN_FLAGS oFlags);
```

Returns

- DRV_ETHPHY_RES_OK - if success, operation complete
- DRV_ETHPHY_RES_PENDING - if function needs to be called again

< 0 - on failure: configuration not supported or some other error

Description

Pointer To Function: typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MDIX_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, TCPIP_ETH_OPEN_FLAGS oFlags);

This type describes a pointer to a function that configures the MDIX mode for the Ethernet PHY. This configuration function is PHY specific and

every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_MDIXConfigure but any name can be used.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_VENDOR_MII_CONFIGURE Type

Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.

File

drv_ethphy.h

C

```
typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MII_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS cFlags);
```

Returns

- DRV_ETHPHY_RES_OK - if success, operation complete
- DRV_ETHPHY_RES_PENDING - if function needs to be called again

< 0 - on failure: configuration not supported or some other error

Description

Pointer To Function: typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MII_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS cFlags);

This type describes a pointer to a function that configures the Ethernet PHY in one of the MII/RMII operation modes. This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_MIIConfigure but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_VENDOR_SMI_CLOCK_GET Type

Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.

File

drv_ethphy.h

C

```
typedef unsigned int (* DRV_ETHPHY_VENDOR_SMI_CLOCK_GET)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle);
```

Returns

The maximum SMI/MIIM clock speed as an unsigned integer.

Description

Pointer to Function: typedef unsigned int (* DRV_ETHPHY_VENDOR_SMI_CLOCK_GET) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle);

This type describes a pointer to a function that returns the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY. This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

This value is PHY specific. All PHYs are requested to support 2.5 MHz.

Traditionally the name used for this function is DRV_EXTPHY_SMIClockGet but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function should not block but return immediately. The function cannot start SMI transactions and cannot use the vendor specific functions to store/retrieve specific data (see Vendor Interface Routines).

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_INDEX_0 Macro

Ethernet PHY driver index definitions.

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_0 0
```

Description

Ethernet PHY Driver Module Index Numbers

These constants provide the Ethernet PHY driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_ETHPHY_Initialize and DRV_ETHPHY_Open routines to identify the driver instance in use.

DRV_ETHPHY_INDEX_1 Macro

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_1 1
```

Description

This is macro DRV_ETHPHY_INDEX_1.

DRV_ETHPHY_INDEX_COUNT Macro

Number of valid Ethernet PHY driver indices.

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_COUNT 1
```

Description

Ethernet PHY Driver Module Index Count

This constant identifies the number of valid Ethernet PHY driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_ETHPHY_LINK_STATUS Enumeration

Defines the possible status flags of PHY Ethernet link.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_LINK_ST_DOWN,
  DRV_ETHPHY_LINK_ST_UP,
  DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE,
  DRV_ETHPHY_LINK_ST_REMOTE_FAULT,
  DRV_ETHPHY_LINK_ST_PDF,
  DRV_ETHPHY_LINK_ST_LP_PAUSE,
  DRV_ETHPHY_LINK_ST_LP_ASM_DIR,
  DRV_ETHPHY_LINK_ST_NEG_TMO,
  DRV_ETHPHY_LINK_ST_NEG_FATAL_ERR
} DRV_ETHPHY_LINK_STATUS;
```

Members

Members	Description
DRV_ETHPHY_LINK_ST_DOWN	No connection to the LinkPartner
DRV_ETHPHY_LINK_ST_UP	Link is up
DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE	LP non negotiation able
DRV_ETHPHY_LINK_ST_REMOTE_FAULT	LP fault during negotiation
DRV_ETHPHY_LINK_ST_PDF	Parallel Detection Fault encountered (when DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE)
DRV_ETHPHY_LINK_ST_LP_PAUSE	LP supports symmetric pause
DRV_ETHPHY_LINK_ST_LP_ASM_DIR	LP supports asymmetric TX/RX pause operation
DRV_ETHPHY_LINK_ST_NEG_TMO	LP not there
DRV_ETHPHY_LINK_ST_NEG_FATAL_ERR	An unexpected fatal error occurred during the negotiation

Description

Ethernet PHY Device Link Status Codes

This enumeration defines the flags describing the status of the PHY Ethernet link.

Remarks

Multiple flags can be set.

DRV_ETHPHY_CONFIG_FLAGS Enumeration

Defines configuration options for the Ethernet PHY.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_CFG_RMII,
  DRV_ETHPHY_CFG_MII,
```

```
  DRV_ETHPHY_CFG_ALTERNATE,
  DRV_ETHPHY_CFG_DEFAULT,
  DRV_ETHPHY_CFG_AUTO
} DRV_ETHPHY_CONFIG_FLAGS;
```

Members

Members	Description
DRV_ETHPHY_CFG_RMII	RMII data interface in configuration fuses.
DRV_ETHPHY_CFG_MII	MII data interface in configuration fuses.
DRV_ETHPHY_CFG_ALTERNATE	Configuration fuses is ALT
DRV_ETHPHY_CFG_DEFAULT	Configuration fuses is DEFAULT
DRV_ETHPHY_CFG_AUTO	Use the fuses configuration to detect if you are RMII/MII and ALT/DEFAULT configuration

Description

Ethernet PHY Configuration Flags

This enumeration defines configuration options for the Ethernet PHY. Used by: DRV_ETHPHY_MIIConfigure, DRV_ETHPHY_INIT structure, DRV_ETHPHY_Setup, Returned by: DRV_ETHPHY_HWConfigFlagsGet

DRV_ETHPHY_OBJECT Structure

Identifies the interface of a Ethernet PHY vendor driver.

File

drv_ethphy.h

C

```
typedef struct {
  DRV_ETHPHY_VENDOR_MII_CONFIGURE miiConfigure;
  DRV_ETHPHY_VENDOR_MDIX_CONFIGURE mdixConfigure;
  DRV_ETHPHY_VENDOR_SMI_CLOCK_GET smiClockGet;
  DRV_ETHPHY_VENDOR_WOL_CONFIGURE wolConfigure;
} DRV_ETHPHY_OBJECT;
```

Members

Members	Description
DRV_ETHPHY_VENDOR_MII_CONFIGURE miiConfigure;	PHY driver function to configure the operation mode: MII/RMII
DRV_ETHPHY_VENDOR_MDIX_CONFIGURE mdixConfigure;	PHY driver function to configure the MDIX mode
DRV_ETHPHY_VENDOR_SMI_CLOCK_GET smiClockGet;	PHY driver function to get the SMI clock rate
DRV_ETHPHY_VENDOR_WOL_CONFIGURE wolConfigure;	PHY driver function to configure the WOL functionality

Description

Ethernet PHY Driver Vendor Object

This data structure identifies the required interface of the Ethernet PHY driver. Any PHY vendor driver has to export this interface.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This object provides vendor specific functionality. Every PHY driver has to expose this vendor specific functionality as part of its interface.

DRV_ETHPHY_VENDOR_WOL_CONFIGURE Type

Pointer to a function to configure the PHY WOL functionality

File

drv_ethphy.h

C

```
typedef void (* DRV_ETHPHY_VENDOR_WOL_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj,
DRV_HANDLE handle, unsigned char bAddr[]);
```

Returns

None

Description

Pointer to Function: typedef void (* DRV_ETHPHY_VENDOR_WOL_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, unsigned char bAddr[]);

This type describes a pointer to a function that configures the PHY WOL functionality of the Ethernet PHY. Configures the WOL of the PHY with a Source MAC address or a 6 byte magic packet mac address.

This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_WOLConfiguration but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

This feature is not currently supported for all PHYs.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_OBJECT_BASE Structure

Identifies the base interface of a Ethernet PHY driver.

File

drv_ethphy.h

C

```
typedef struct DRV_ETHPHY_OBJECT_BASE_TYPE {
  SYS_MODULE_OBJ (* DRV_ETHPHY_Initialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
  void (* DRV_ETHPHY_Reinitialize)(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
  void (* DRV_ETHPHY_Deinitialize)(SYS_MODULE_OBJ object);
  SYS_STATUS (* DRV_ETHPHY_Status)(SYS_MODULE_OBJ object);
  void (* DRV_ETHPHY_Tasks)(SYS_MODULE_OBJ object);
  DRV_HANDLE (* DRV_ETHPHY_Open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
  void (* DRV_ETHPHY_Close)(DRV_HANDLE handle);
  DRV_ETHPHY_CLIENT_STATUS (* DRV_ETHPHY_ClientStatus)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_ClientOperationResult)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_ClientOperationAbort)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIRead)(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int
phyAdd);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIWrite)(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int
phyAdd, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStart)(DRV_HANDLE handle, unsigned int rIx);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStop)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStatusGet)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanDataGet)(DRV_HANDLE handle, uint16_t* pScanRes);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIStatus)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIClockSet)(DRV_HANDLE handle, uint32_t hostClock, uint32_t maxSMIClock);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_PhyAddressGet)(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex,
int* pPhyAddress);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_Setup)(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS*
pSetupFlags);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_RestartNegotiation)(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX
portIndex);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_HWConfigFlagsGet)(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_NegotiationIsComplete)(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX
```

```
portIndex, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_NegotiationResultGet)(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX
portIndex, DRV_ETHPHY_NEGOTIATION_RESULT* pNegResult);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_LinkStatusGet)(DRV_HANDLE handle, DRV_ETHPHY_INTERFACE_INDEX portIndex,
DRV_ETHPHY_LINK_STATUS* pLinkStat, bool refresh);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_Reset)(DRV_HANDLE handle, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorDataGet)(DRV_HANDLE handle, uint32_t* pVendorData);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorDataSet)(DRV_HANDLE handle, uint32_t vendorData);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIReadStart)(DRV_HANDLE handle, uint16_t rIx, int phyAddress);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIReadResultGet)(DRV_HANDLE handle, uint16_t* pSmiRes);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIWriteStart)(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int
phyAddress);
} DRV_ETHPHY_OBJECT_BASE;
```

Description

Ethernet PHY Driver Base Object

This data structure identifies the required interface of the Ethernet PHY driver. Any dynamic PHY driver has to export this interface.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This object provides the base functionality. Every dynamic PHY driver has to expose this basic functionality as part of its interface.

See above the description of each function that's part of the base PHY driver.

DRV_ETHPHY_RESET_FUNCTION Type

Pointer to a function to perform an additional PHY reset

File

drv_ethphy.h

C

```
typedef void (* DRV_ETHPHY_RESET_FUNCTION)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj);
```

Returns

None

Description

Pointer to Function: typedef void (* DRV_ETHPHY_RESET_FUNCTION) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj);

This type describes a pointer to a function that is called by the driver before starting the detection and initialization process to the PHY - as a result of the DRV_ETHPHY_Setup call.

Remarks

The PHY driver will call this function as part of its detection and initialization procedure. It can be used for implementing extra steps that the user needs, before the driver starts talking to the PHY. For example, if a hard reset needs to be applied to the PHY.

The function should be short and not block. It is meant just for short I/O operations, not for lengthy processing.

Preconditions

None

DRV_ETHPHY_RESULT Enumeration

Defines the possible results of Ethernet operations that can succeed or fail

File

drv_ethphy.h

C

```
typedef enum {
} DRV_ETHPHY_RESULT;
```

Description

Ethernet PHY Driver Operation Result *

PHY Driver Operation Result Codes

This enumeration defines the possible results of any of the PHY driver operations that have the possibility of failing. This result should be checked to ensure that the operation achieved the desired result.

DRV_ETHPHY_USE_DRV_MIIM Macro

Defines the way the PHY driver accesses the MIIM bus to communicate with the PHY.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_USE_DRV_MIIM true
```

Description

Ethernet MIIM access configuration

Defines the way the PHY driver accesses the MIIM bus to communicate with the PHY:

- either using direct access to the ETH plibs
- using the MIIM driver - preferred way

Remarks

Using the MIIM driver to perform MIIM bus operations is more versatile and preferred.

DRV_ETHPHY_INTERFACE_INDEX Enumeration

Defines the index type for a PHY interface.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_INF_IDX_ALL_EXTERNAL,
  DRV_ETHPHY_INF_IDX_PORT_0,
  DRV_ETHPHY_INF_IDX_PORT_1,
  DRV_ETHPHY_INF_IDX_PORT_2,
  DRV_ETHPHY_INF_IDX_PORT_3,
  DRV_ETHPHY_INF_IDX_PORT_4,
  DRV_ETHPHY_INF_IDX_PORT_5
} DRV_ETHPHY_INTERFACE_INDEX;
```

Members

Members	Description
DRV_ETHPHY_INF_IDX_ALL_EXTERNAL	All External Interfaces
DRV_ETHPHY_INF_IDX_PORT_0	Port 0 interface
DRV_ETHPHY_INF_IDX_PORT_1	Port 1 interface
DRV_ETHPHY_INF_IDX_PORT_2	Port 2 interface
DRV_ETHPHY_INF_IDX_PORT_3	Port 3 interface
DRV_ETHPHY_INF_IDX_PORT_4	Port 4 interface
DRV_ETHPHY_INF_IDX_PORT_5	Port 5 interface

Description

Ethernet PHY Interface Index

This enumeration defines the index type supported by the PHY Used by: DRV_ETHPHY_PhyAddressGet, DRV_ETHPHY_RestartNegotiation, DRV_ETHPHY_NegotiationIsComplete, DRV_ETHPHY_LinkStatusGet

DRV_ETHPHY_INTERFACE_TYPE Enumeration

Defines the type of interface a PHY supports.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_INF_TYPE_EXTERNAL,
  DRV_ETHPHY_INF_TYPE_INTERNAL,
  DRV_ETHPHY_INF_TYPE_NOT_SUPPORTED
} DRV_ETHPHY_INTERFACE_TYPE;
```

Members

Members	Description
DRV_ETHPHY_INF_TYPE_EXTERNAL	External Interface
DRV_ETHPHY_INF_TYPE_INTERNAL	Internal Interface
DRV_ETHPHY_INF_TYPE_NOT_SUPPORTED	Not Supported

Description

Ethernet PHY Interface Type

This enumeration defines the type of interface supported by the PHY Returned by: DRV_ETHPHY_GetInterfaceType

Files

Files

Name	Description
drv_ethphy.h	Ethernet ETHPHY Device Driver Interface File
drv_ethphy_config.h	Ethernet PHY driver configuration definitions template.

Description

This section lists the source and header files used by the Ethernet PHY Driver Library.

drv_ethphy.h

Ethernet ETHPHY Device Driver Interface File

Enumerations

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	DRV_ETHPHY_CONFIG_FLAGS	Defines configuration options for the Ethernet PHY.
	DRV_ETHPHY_INTERFACE_INDEX	Defines the index type for a PHY interface.
	DRV_ETHPHY_INTERFACE_TYPE	Defines the type of interface a PHY supports.
	DRV_ETHPHY_LINK_STATUS	Defines the possible status flags of PHY Ethernet link.
	DRV_ETHPHY_RESULT	Defines the possible results of Ethernet operations that can succeed or fail

Functions

	Name	Description
	DRV_ETHPHY_ClientOperationAbort	Aborts a current client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationResult	Gets the result of a client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags. **Implementation:** Dynamic
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver. **Implementation:** Dynamic

	DRV_ETHPHY_LinkStatusGet	Returns the current link status. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationResultGet	Returns the result of a completed negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_ETHPHY_PhyAddressGet	Returns the PHY address. **Implementation:** Dynamic
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY. **Implementation:** Dynamic
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link. **Implementation:** Dynamic
	DRV_ETHPHY_Setup	Initializes Ethernet PHY configuration and set up procedure. **Implementation:** Dynamic
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock. **Implementation:** Dynamic
	DRV_ETHPHY_SMIRead	Initiates a SMI/MIIM read transaction. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStart	Starts the scan of a requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStop	Stops the scan of a previously requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIStatus	Returns the current status of the SMI/MIIM interface. **Implementation:** Dynamic
	DRV_ETHPHY_SMIWrite	Initiates a SMI/MIIM write transaction. **Implementation:** Dynamic
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataGet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataSet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadResultGet	Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadStart	Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIWriteStart	Starts a vendor SMI write transfer. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.

Structures

	Name	Description
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.
	DRV_ETHPHY_OBJECT_BASE_TYPE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_NEGOTIATION_RESULT	Contains all the data necessary to get the Ethernet PHY negotiation result
	DRV_ETHPHY_OBJECT	Identifies the interface of a Ethernet PHY vendor driver.
	DRV_ETHPHY_OBJECT_BASE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_SETUP	Contains all the data necessary to set up the Ethernet PHY device.

Types

	Name	Description
	DRV_ETHPHY_RESET_FUNCTION	Pointer to a function to perform an additional PHY reset
	DRV_ETHPHY_VENDOR_MDIX_CONFIGURE	Pointer to function that configures the MDIX mode for the Ethernet PHY.
	DRV_ETHPHY_VENDOR_MII_CONFIGURE	Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.
	DRV_ETHPHY_VENDOR_SMI_CLOCK_GET	Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.
	DRV_ETHPHY_VENDOR_WOL_CONFIGURE	Pointer to a function to configure the PHY WOL functionality

Description

Ethernet ETHPHY Device Driver Interface

The Ethernet ETHPHY device driver provides a simple interface to manage an Ethernet ETHPHY peripheral using MIIM (or SMI) interface. This file defines the interface definitions and prototypes for the Ethernet ETHPHY driver.

File Name

drv_ethphy.h

Company

Microchip Technology Inc.

drv_ethphy_config.h

Ethernet PHY driver configuration definitions template.

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_NEG_DONE_TMO	Value of the PHY negotiation complete time out as per IEEE 802.3 spec.
	DRV_ETHPHY_NEG_INIT_TMO	Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHPHY_RESET_CLR_TMO	Value of the PHY Reset self clear time out as per IEEE 802.3 spec.
	DRV_ETHPHY_USE_DRV_MIIM	Defines the way the PHY driver accesses the MIIM bus to communicate with the PHY.

Description

Ethernet PHY Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_ethphy_config.h

Company

Microchip Technology Inc.

Flash Driver Library

This section describes the Flash Driver Library.

Introduction

The Flash Driver Library provides functions that allow low-level interface with the on-chip Flash.

Description

Through MHC, this driver provides low-level functions for writing and erasing sections of the Flash memory.

Flash Program Memory

The Flash Program Memory is readable, writeable, and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete, restricting memory access, and therefore preventing code execution. This is controlled by an internal programming timer.

Library Interface

Functions

	Name	Description
	DRV_FLASH_ErasePage	Erases a page of Flash. **Implementation:** Static
	DRV_FLASH_GetPageSize	Returns the size in bytes of a single "Page" which can be erased in the flash. **Implementation:** Static
	DRV_FLASH_GetRowSize	Returns the size in bytes of a single "Row" which can be written to the flash. **Implementation:** Static
	DRV_FLASH_Initialize	Initializes the Flash instance for the specified driver index. **Implementation:** Static
	DRV_FLASH_IsBusy	Returns true if the Flash device is still busy writing or is erasing. **Implementation:** Static
	DRV_FLASH_Open	Initializes a channel to the appropriate flash device.
	DRV_FLASH_WriteQuadWord	Writes four 4-byte words to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteRow	Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteWord	Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr. **Implementation:** Static

Data Types and Constants

	Name	Description
	DRV_FLASH_INDEX_0	FLASH driver index definitions
	DRV_FLASH_PAGE_SIZE	Specifies the FLASH Driver Program Page Size in bytes.
	DRV_FLASH_ROW_SIZE	Specifies the FLASH Driver Program Row Size in bytes.

Description

This section describes the Application Programming Interface (API) functions of the Flash Driver Library.

Refer to each section for a detailed description.

Functions

DRV_FLASH_ErasePage Function

Erases a page of Flash.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_ErasePage(const DRV_HANDLE handle, uint32_t flashAddr);
```

Returns

None.

Description

This function starts the process of erasing a page of Flash. It does not wait for the erase operation to be done. That is left to the user. It does not verify that the erase was successful. That is left to the user. It always erases a single page. The size of a page in bytes will vary by device. It will be available in the DRV_FLASH_PAGE_SIZE parameter.

Remarks

Most devices will be running for code stored in the Flash. This means that any erases of the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
DRV_FLASH_Erase_Page(handle, flashAddr);
```

Function

void DRV_FLASH_Erase_Page(uint32_t flashAddr);

DRV_FLASH_GetPageSize Function

Returns the size in bytes of a single "Page" which can be erased in the flash.

Implementation: Static

File

drv_flash.h

C

```
uint32_t DRV_FLASH_GetPageSize(const DRV_HANDLE handle);
```

Returns

None.

Description

This function allows the user to get the size of a flash Page.

Remarks

None.

Preconditions

None

Function

uint32_t DRV_FLASH_GetPageSize(const DRV_HANDLE handle)

DRV_FLASH_GetRowSize Function

Returns the size in bytes of a single "Row" which can be written to the flash.

Implementation: Static

File

drv_flash.h

C

```
uint32_t DRV_FLASH_GetRowSize(const DRV_HANDLE handle);
```

Returns

None.

Description

This function allows the user to get the size of a flash Row.

Remarks

None.

Preconditions

None

Function

uint32_t DRV_FLASH_GetRowSize(const DRV_HANDLE handle)

DRV_FLASH_Initialize Function

Initializes the Flash instance for the specified driver index.

Implementation: Static

File

drv_flash.h

C

```
SYS_MODULE_OBJ DRV_FLASH_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the Flash Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This function must be called before any other Flash function is called. This function should only be called once during system initialization.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_FLASH_Initialize(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
)

DRV_FLASH_IsBusy Function

Returns true if the Flash device is still busy writing or is erasing.

Implementation: Static

File

drv_flash.h

C

```
bool DRV_FLASH_IsBusy(const DRV_HANDLE handle);
```

Returns

- true - Indicates the Flash is busy
- false - Indicates the Flash is not busy

Description

This function checks whether the process of programming a Word into the Flash is still operating.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

None.

Example

```
flashAddr = 0x9d008000;
sourceData = 0x12345678;
DRV_FLASH_Write_Word(flashAddr,  sourceData);
DRV_FLASH_IsBusy( void );
```

Function

bool DRV_FLASH_IsBusy(void)

DRV_FLASH_Open Function

Initializes a channel to the appropriate flash device.

File

drv_flash.h

C

```
DRV_HANDLE DRV_FLASH_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

Handle for future calls to the driver's operations.

Preconditions

None

Function

DRV_HANDLE DRV_FLASH_Open(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

DRV_FLASH_WriteQuadWord Function

Writes four 4-byte words to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteQuadWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t * sourceData);
```

Returns

None.

Description

This function starts the process of programming a word into the Flash. It does not wait for the write operation to be done, which is left to the user. It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
sourceData[4] = {0x12345678,0x9ABCDEF0,0x55AAAA55,0x11111111};
DRV_FLASH_WriteQuadWord(handle, flashAddr,  sourceData);
```

Function

void DRV_FLASH_WriteQuadWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

DRV_FLASH_WriteRow Function

Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteRow(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData);
```

Returns

None.

Description

This function starts the process of programming a buffer into the Flash. It does not wait for the write operation to be done, which is left to the user. It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. The memory pointed to by sourceData must be valid memory for at least DRV_FLASH_ROW_SIZE bytes. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
```

```
uint32_t dataStore[DRV_FLASH_ROW_SIZE] = {0,1,2,3,4,5};
DRV_FLASH_Write_Row( const DRV_HANDLE handle, flashAddr,  dataStore);
```

Function

void DRV_FLASH_WriteRow(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

DRV_FLASH_WriteWord Function

Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData);
```

Returns

None.

Description

This function starts the process of programming a Word into the Flash. It does not wait for the write operation to be done, which is left to the user. It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
sourceData = 0x12345678;
DRV_FLASH_WriteWord(handle, flashAddr,  sourceData);
```

Function

void DRV_FLASH_WriteWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

Data Types and Constants

DRV_FLASH_INDEX_0 Macro

FLASH driver index definitions

File

drv_flash.h

C

```
#define DRV_FLASH_INDEX_0 0
```

Description

These constants provide FLASH driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_FLASH_Initialize and DRV_FLASH_Open routines to identify the driver instance in use.

Section

Constants

****

****

****

Driver FLASH Module Index

DRV_FLASH_PAGE_SIZE Macro

Specifies the FLASH Driver Program Page Size in bytes.

File

drv_flash.h

C

```
#define DRV_FLASH_PAGE_SIZE (NVM_PAGE_SIZE)
```

Description

FLASH Driver Program Page Size.

This definition specifies the FLASH Driver Program Page Size in bytes. This parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

DRV_FLASH_ROW_SIZE Macro

Specifies the FLASH Driver Program Row Size in bytes.

File

drv_flash.h

C

```
#define DRV_FLASH_ROW_SIZE (NVM_ROW_SIZE)
```

Description

FLASH Driver Program Row Size.

This definition specifies the FLASH Driver Program Row Size in bytes. This parameter is device specific and is obtained from the device specific processor header file. The Program Row Size is the maximum block size that can be programmed in one program operation.

Remarks

None

Files

Files

Name	Description
drv_flash.h	Flash Driver interface declarations for the static single instance driver.

Description

drv_flash.h

Flash Driver interface declarations for the static single instance driver.

Functions

	Name	Description
	DRV_FLASH_ErasePage	Erases a page of Flash. **Implementation:** Static
	DRV_FLASH_GetPageSize	Returns the size in bytes of a single "Page" which can be erased in the flash. **Implementation:** Static
	DRV_FLASH_GetRowSize	Returns the size in bytes of a single "Row" which can be written to the flash. **Implementation:** Static
	DRV_FLASH_Initialize	Initializes the Flash instance for the specified driver index. **Implementation:** Static
	DRV_FLASH_IsBusy	Returns true if the Flash device is still busy writing or is erasing. **Implementation:** Static
	DRV_FLASH_Open	Initializes a channel to the appropriate flash device.
	DRV_FLASH_WriteQuadWord	Writes four 4-byte words to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteRow	Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteWord	Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr. **Implementation:** Static

Macros

	Name	Description
	DRV_FLASH_INDEX_0	FLASH driver index definitions
	DRV_FLASH_PAGE_SIZE	Specifies the FLASH Driver Program Page Size in bytes.
	DRV_FLASH_ROW_SIZE	Specifies the FLASH Driver Program Row Size in bytes.

Description

Flash Driver Interface Declarations for Static Single Instance Driver

The Flash device driver provides a simple interface to manage the Flash Controller on Microchip microcontrollers. This file defines the interface Declarations for the Flash driver.

Remarks

Static interfaces incorporate the driver instance number within the names of the routines, eliminating the need for an object ID or object handle.

Static single-open interfaces also eliminate the need for the open handle.

File Name

drv_flash.h

Company

Microchip Technology Inc.

Ethernet GMAC Driver Library

This section describes the Ethernet MAC Driver Library.

Introduction

This library provides a driver-level abstraction of the on-chip Ethernet Controller found on many PIC32 devices. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module help for details.

The "Host-To-Network" layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section. An external Ethernet "PHY" provides the Physical layer, providing conversion between the digital and analog.

Description

The Ethernet Media Access Controller (GMAC) module implements a 10/100 Mbps Ethernet MAC, compatible with the IEEE 802.3 standard. The GMAC can operate in either half or full duplex mode at all supported speeds.

Embedded Characteristics

- Compatible with IEEE Standard 802.3
- 10, 100 Mbps operation
- Full and half duplex operation at all supported speeds of operation
- Statistics Counter Registers for RMON/MIB
- MII interface to the physical layer
- Integrated physical coding
- Direct memory access (DMA) interface to external memory
- Support for 6 priority queues in DMA
- 8 KB transmit RAM and 4 KB receive RAM
- Programmable burst length and endianism for DMA
- Interrupt generation to signal receive and transmit completion, errors or other events
- Automatic pad and cyclic redundancy check (CRC) generation on transmitted frames
- Automatic discard of frames received with errors
- Receive and transmit IP, TCP and UDP checksum offload. Both IPv4 and IPv6 packet types supported
- Address checking logic for four specific 48-bit addresses, four type IDs, promiscuous mode, hash matching of unicast and multicast destination addresses and Wake-on-LAN
- Management Data Input/Output (MDIO) interface for physical layer management
- Support for jumbo frames up to 10240 Bytes
- Full duplex flow control with recognition of incoming pause frames and hardware generation of transmitted pause frames
- Half duplex flow control by forcing collisions on incoming frames
- Support for 802.1Q VLAN tagging with recognition of incoming VLAN and priority tagged frames
- Support for 802.1Qbb priority-based flow control
- Programmable Inter Packet Gap (IPG) Stretch
- Recognition of IEEE 1588 PTP frames
- IEEE 1588 time stamp unit (TSU)
- Support for 802.1AS timing and synchronization
- Supports 802.1Qav traffic shaping on two highest priority queues

Using the Library

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

This topic describes the basic architecture and functionality of the Ethernet MAC driver and is meant for advanced users or TCP/IP stack driver developers.

Interface Header File: `drv_gmac.h`

The interface to the Ethernet MAC library is defined in the `drv_gmac.h` header file, which is included by the MPLAB Harmony TCP/IP stack.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the Ethernet GMAC Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

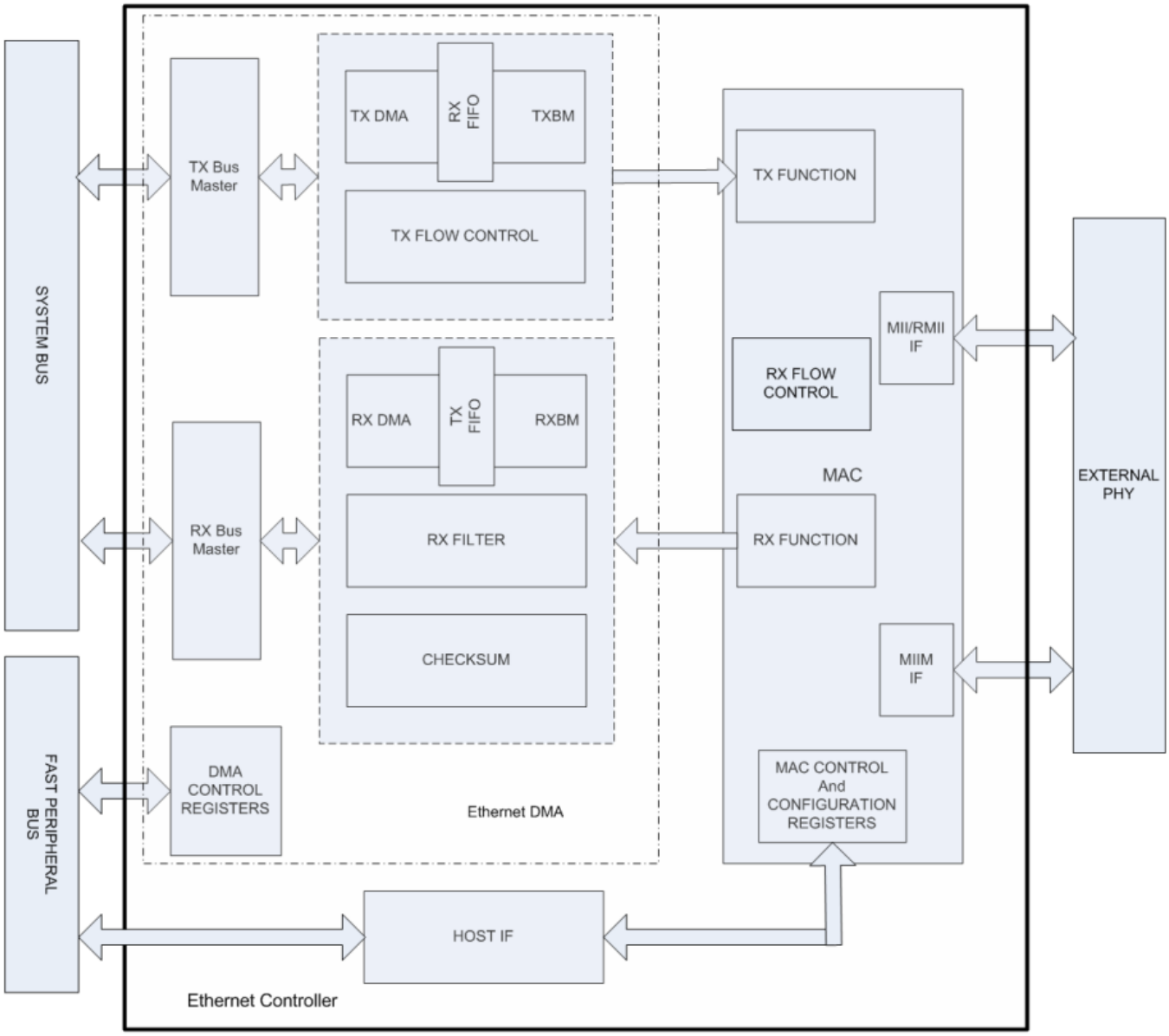
Description

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external Ethernet PHY chip. The PHY chip provides a digital-analog interface as part of the Physical Layer and the controller provides the Media Access Controller (MAC) layer above the PHY.

As shown in Figure 1, the Ethernet Controller consists of the following modules:

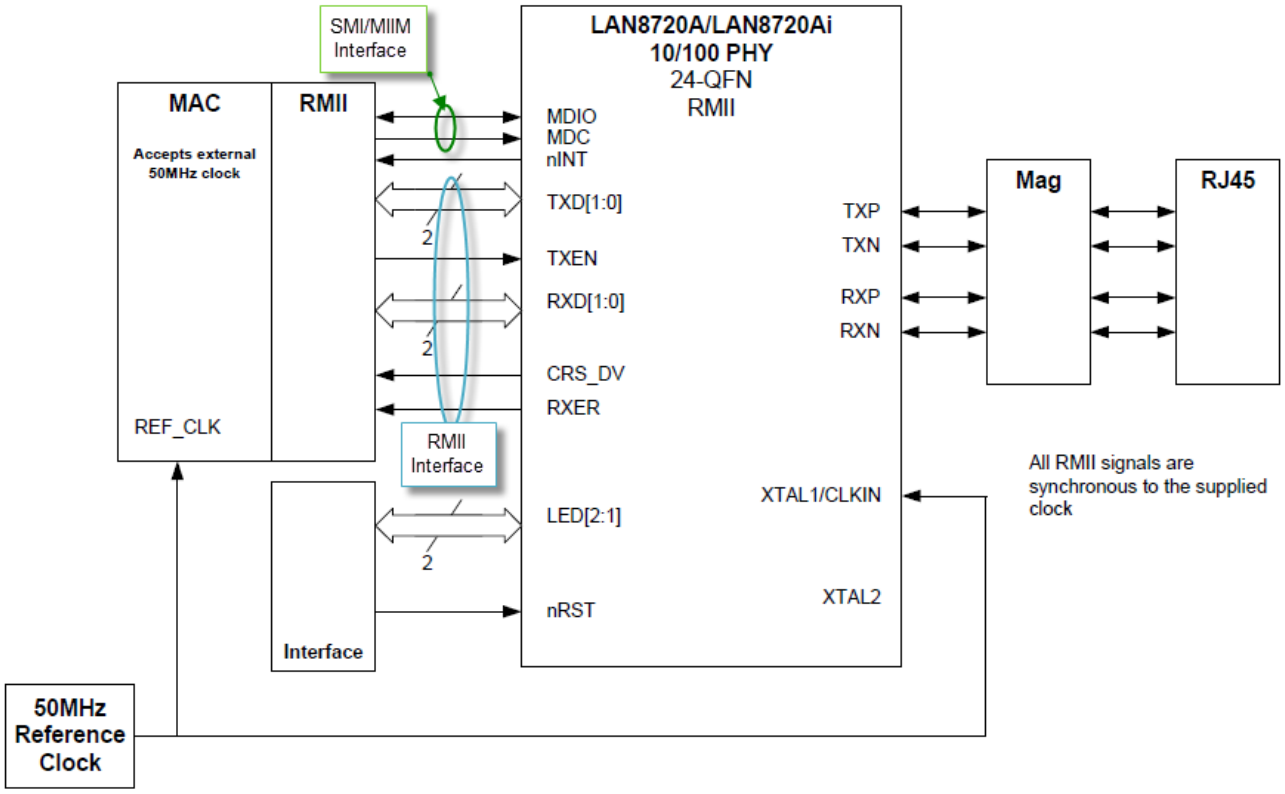
- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block: Responsible for control of the transmission of PAUSE frames. (Reception of PAUSE frames is handled within the MAC.)
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected
- TX DMA/TX Buffer Management Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX Buffer Management Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Figure 1: Ethernet Controller Block Diagram



For completeness, we also need to look at the interface diagram of a representative Ethernet PHY. As shown in Figure 2, the PHY has two interfaces, one for configuring and managing the PHY (SMI/MIIM) and another for transmit and receive data (RMII or MII). The SMI/MIIM interface is the responsibility of the Ethernet PHY Driver Library. When setting up the Ethernet PHY, this Ethernet driver calls primitives from the Ethernet PHY Driver library. The RMII/MII data interface is the responsibility of the Ethernet MAC Driver Library (this library).

Figure 2: Ethernet PHY Interfaces



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system. Refer to the TCP/IP Stack Library MAC Driver Module help for the interface that the Ethernet driver has to implement in a MPLAB Harmony system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet GMAC Driver Library.

Library Interface Section	Description
Client Level Functions	Open, Close, Initialize, Reinitialize, and Deinitialize functions to support the TCP/IP Stack. Plus link status and power options.
Receive Functions	Receive routines.
Transmit Functions	Transmit routines.
Event Functions	Ethernet event support routines.
Other Functions	Additional routines.
Data Types and Constants	Typedefs and #defines.

Configuring the Library

The configuration of the Ethernet MAC driver is done as part of the MPLAB Harmony TCP/IP Stack configuration and is based on the `system_config.h` file, which may include the `tcpip_mac_config.h`. See the TCP/IP Stack Library MAC Driver Module help file for configuration options.

This header file contains the configuration selection for the Ethernet GMAC Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Ethernet GMAC Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/gmac`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_gmac.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_gmac.c`	PIC32 internal Ethernet driver virtual GMAC implementation file.
`/src/dynamic/drv_gmac_lib.c`	PIC32 internal Ethernet driver controller implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Ethernet MAC Driver Library depends on the following modules:

- Ethernet PHY Driver Library
- Interrupt System Service Library
- Timer System Service Library
- Ethernet Peripheral Library

Library Interface

This section lists the interface routines, data types, constants and macros for the library.

a) Client Level Functions

b) Receive Functions

c) Transmit Functions

d) Event Functions

e) Other Functions

f) Data Types and Constants

Files

Files

Name	Description
drv_gmac.h	This is file drv_gmac.h.

Description

This section lists the source and header files used by the Ethernet MAC Driver Library.

drv_gmac.h

This is file drv_gmac.h.

I2C Driver Library Help

This section describes the I2C Driver Library.

Introduction

This library provides an interface to manage the data transfer operations using the I2C module on the Microchip family of microcontrollers.

Description

The driver communicates using the concept of *transactions*. In instances where the I2C operates in Master mode, the driver sends the start signal, followed by a slave device address (including a Read/Write bit), followed by a number of bytes written to or read from the slave. The *transaction* is completed by sending the stop signal. When the driver operates in the Slave mode, it will either read data or write data to the master.

This driver library provides application ready routines to read and write data using the I2C protocol, thus minimizing developer's awareness of the working of the I2C protocol.

- Provides read/write and buffer data transfer models
- Supports interrupt and Polled modes of operation
- Support multi-client and multi-instance operation
- Provides data transfer events
- Supports blocking and non-blocking operation
- Supports baud rate setting
- Supports bit bang mode.

Using the Library

This topic describes the basic architecture of the I2C Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_i2c.h`

The interface to the I2C Driver Library is defined in the `drv_i2c.h` header file. Any C language source (`.c`) file that uses the I2C Driver Library should include `drv_i2c.h`.

Library File: The I2C Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The I2C Driver Library provides the low-level abstraction of the I2C module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the I2C Driver Library interface.

Description

The I2C Driver Library features routines to perform two functions, driver maintenance and data transfer:

Driver Maintenance

The Driver initialization routines allow the application to initialize the driver. The initialization data configures the I2C module as a Master or a Slave and sets the necessary parameters required for operation in the particular mode. The driver must be initialized before it can be used by the application. After the end of operation, the driver can be deinitialized.

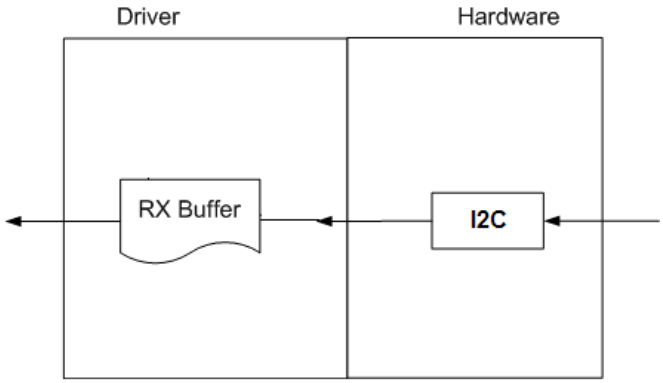
Data Transfer

Data transfer is accomplished by separate Write and Read functions through a data buffer. The read and write function makes the user transparent to the internal working of the I2C protocol. The user can use callback mechanisms or use polling to check status of transfer.

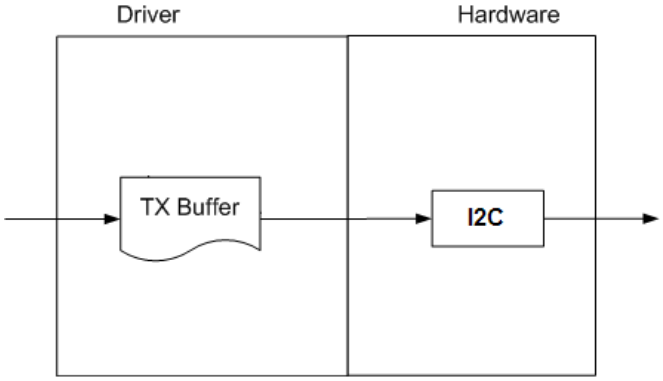
The following diagrams illustrate the model used by the I2C Driver for transmitter and receiver.

Note: The driver can be configured to use either the SOC peripheral, or in Bit Bang mode. Bit Bang mode uses CPU resources to process the data bits onto or off of the I2C I/O pins. Be aware that and I2C driver configured this way uses a large amount of CPU resources.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2C Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This section provides information on system access.

Description

System Access

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the I2C module would be initialized with the following configuration settings (either passed dynamically at run-time using DRV_I2C_INIT or by using initialization overrides) that are supported by the specific I2C device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., I2C_ID_2)
- Master or Slave mode of operation and their associated parameters
- Defining the respective interrupt sources for Master, Slave, and Error Interrupt

The DRV_I2C_Initialize API returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_I2C_Deinitialize, DRV_I2C_Status, and DRV_I2C_Tasks.

Note: The system initialization settings, only affect the instance of the peripheral that is being initialized.

Example:

```
DRV_I2C_INIT        i2c_init_data;
SYS_MODULE_OBJ      objectHandle;

i2c_init_data.i2cId = DRV_I2C_PERIPHERAL_ID_IDX0,
i2c_init_data.i2cMode = DRV_I2C_MODE_MASTER,
OR
i2c_init_data.i2cMode = DRV_I2C_MODE_SLAVE,

/*    Master mode parameters */
i2c_init_data.baudRate = 100000,
i2c_init_data.busspeed = DRV_I2C_SLEW_RATE_CONTROL_IDX0,
i2c_init_data.buslevel = DRV_I2C_SMBus_SPECIFICATION_IDX0,

/*    Master mode parameters */
i2c_init_data.addWidth = DRV_I2C_7BIT_SLAVE,
i2c_init_data.reservedaddenable  = false,
i2c_init_data.generalcalladdress = false,
i2c_init_data.slaveaddvalue = 0x0060,

//interrupt sources
i2c_init_data.mstrInterruptSource = INT_SOURCE_I2C_2_MASTER,
i2c_init_data.slaveInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.errInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.queueSize = 1,

/* callback for Master (Master mode can use callbacks if needed) */
i2c_init_data.operationStarting = NULL,

/* Slave mode callbacks needed */
i2c_init_data.operationStarting = APP_I2CSlaveFunction,

objectHandle = DRV_I2C_Initialize(DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData)
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
   // Handle error
}
```

Since the I2C bus is controlled by the Master, the Slave should respond to a read or write request whenever the Master makes the request. Thus, the slave does not have driver states like the Master. The operation of the I2C Driver when used in Slave mode is handled using callbacks. The callback, OperationStarting, must be configured during system initialization when in Slave mode. This callback is provided so that the application can respond appropriately when a read or write request is received from the Master.

Client Access

This section provides information on client access.

Description

For the application to start using an instance of the module, it must call the DRV_I2C_Open function. This provides the configuration required to open the I2C instance for operation. If the driver is deinitialized using the function DRV_I2C_Deinitialize, the application must call the DRV_I2C_Open function again to set up the instance of the I2C.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

After a client instance is opened, DRV_I2C_ClientSetup can be called to set up client-specific parameters. In I2C Slave mode, this is used to set-up the IRQ logic so that the slave can toggle this line to request Master to send a Read command.

As during initialization, when the I2C module operates in the Slave mode, only the Master can terminate a transaction with the Slave. In this case, the driver provides a callback to the application after the reception of each byte from the Master or after transmission of a byte to the Master.

Example:

```
/* I2C Driver Handle  */
DRV_HANDLE  drvI2CHandle;

/* Open the I2C Driver */
appData.drvI2CHandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

if (drvI2CHandle != DRV_HANDLE_VALID)
{
     //Client cannot open instance
}
```

Client Transfer

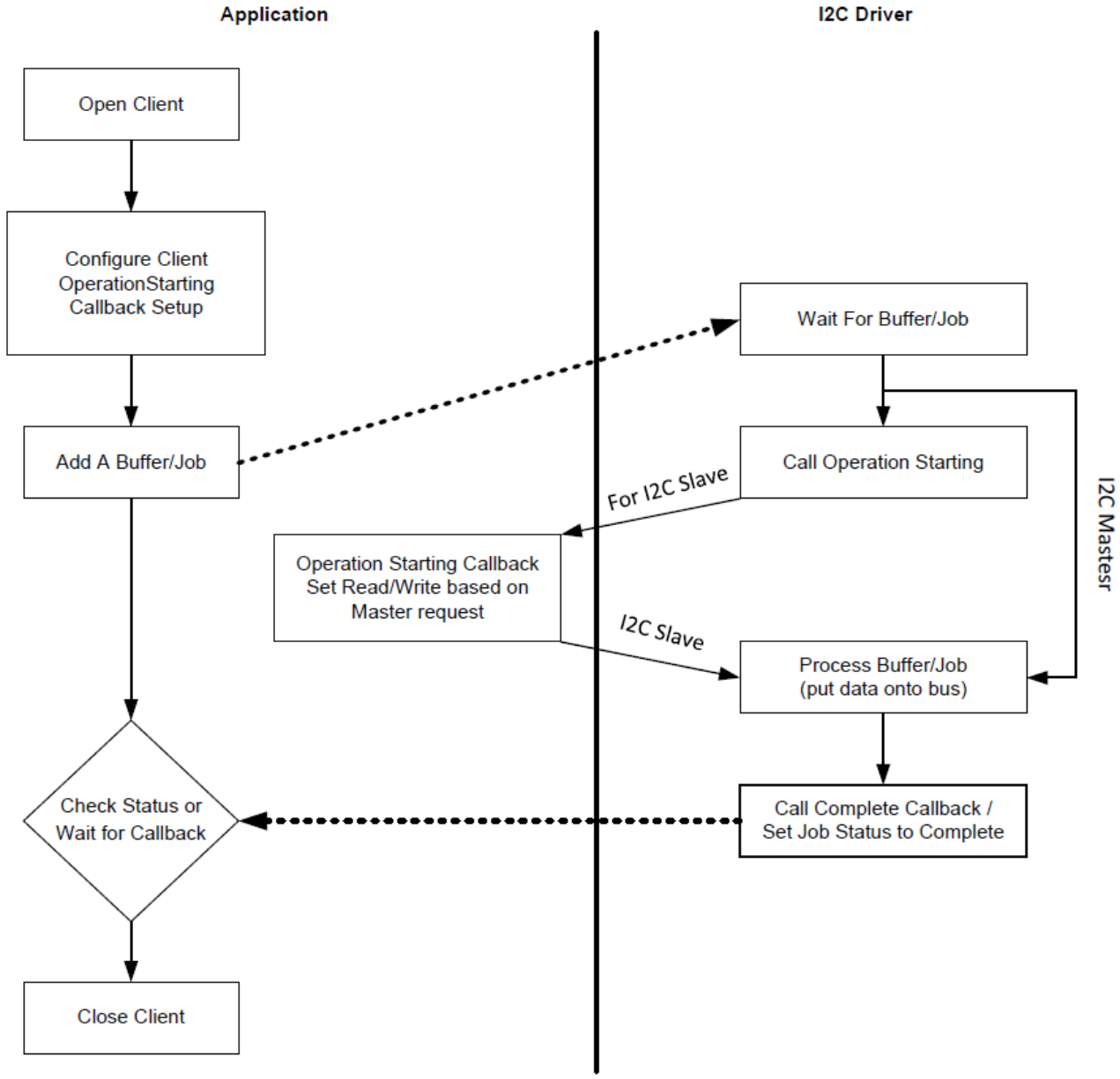
This section provides information on client transfer functionality.

Description

Core Functionality

Client basic functionality provides an extremely basic interface for the driver operation.

The following diagram illustrates the byte/word model used for the data transfer.



Client Data Transfer Functionality

Applications using the I2C driver need to perform the following:

1. The system should have completed necessary initialization and the DRV_I2C_Tasks should either be running in polled environment, or in an interrupt environment.
2. Open the driver using DRV_I2C_Open with the necessary intent.
3. Add a buffer using the DRV_I2C_Receive, DRV_I2C_Transmit, and DRV_I2C_TransmitThenReceive functions. An optional callback can be provided that will be called when the buffer/job is complete using DRV_I2C_BufferEventHandlerSet.
4. Check for the current transfer status using DRV_I2C_TransferStatusGet or wait for the callback to be called with buffer transfer status set to DRV_I2C_BUFFER_EVENT_COMPLETE or DRV_I2C_BUFFER_EVENT_ERROR.
5. Buffer status DRV_I2C_BUFFER_EVENT_COMPLETE implies that the I2C transaction has been completed without any errors. Buffer status DRV_I2C_BUFFER_EVENT_ERROR indicates that the I2C transaction was aborted and the entire contents of the buffer were not transferred.
6. In Master mode, common cases for DRV_I2C_BUFFER_EVENT_ERROR to be set are:

- Slave is non-operational
- Slave is performing an internal operation and cannot accept any more I2C messages from the Master until the operation completes. In such a case, if the Master tries to address the Slave and is attempting to transfer data, the Slave NACKs the transfer. This will result in the Master prematurely terminating the transaction and setting the DRV_I2C_BUFFER_EVENT_FLAG. In the application level, the Master can continuously attempt to send the transaction until transfer status changes from for DRV_I2C_BUFFER_EVENT_ERROR to DRV_I2C_BUFFER_EVENT_COMPLETE. This will in effect perform the so-called "Acknowledge Polling". An example of a Slave device that depicts this behavior is an EEPROM.

7. The client will be able to close the driver using DRV_I2C_Close when required.

Example:

```
/*  This example demonstrates the I2C driver setup of one instance of I2C acting
    as a Master to another instance of the I2C Driver acting as a Slave.
    In the Slave initialization data structure in system_init.c, the member
    operationStarting should be assigned a function pointer. This function will
    be called when the Slave receives an address match. Based on the R/W bit in
    the address, the transmit or receive function will be called by the Slave
    (e.g., .operationStarting = APP_SlaveDataforMaster) */

SYS_MODULE_OBJ i2cMasterObject;

SYS_MODULE_OBJ i2cSlaveObject;


/* function prototype of callback function */
void I2CMasterOpStatusCb ( DRV_I2C_BUFFER_EVENT event,
                           DRV_I2C_BUFFER_HANDLE bufferHandle,
                           uintptr_t context);

int main( void )
{
    while ( 1 )
    {
        appTask ();
    }
}

void appTask ()
{
    #define MY_BUFFER_SIZE             5
    #define RTCC_SLAVE_ADDRESS        0xDE

    /* initialize slave address value */
    unsigned char address             =    RTCC_SLAVE_ADDRESS;

    /*Initialize myBuffer with MY_BUFFER_SIZE bytes of valid data */
    char myBuffer[MY_BUFFER_SIZE] = { 11, 22, 33, 44, 55};
    unsigned int numBytes;

    DRV_HANDLE drvI2CMasterHandle;        //Returned from DRV_I2C_Open for I2C Master
    DRV_I2C_BUFFER_HANDLE bufHandle_M1;   //Returned from calling a Data Transfer function
    uintptr_t i2cOpStatus;                //Operation status of I2C operation returned from callback

    DRV_HANDLE drvI2CSlaveHandle;         //Returned from DRV_I2C_Open for I2C Slave
    DRV_I2C_BUFFER_HANDLE bufHandle_S1;   //Returned from calling a Data Transfer function
    DRV_I2C_BUFFER_HANDLE bufHandle_S2;   //Returned from calling a Data Transfer function



    while( 1 )
    {
        switch( state )
        {
            case APP_STATE_INIT:
            {
                /* Initialize the Master I2C Driver */
                i2cMasterObject = DRV_I2C_Initialize( DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData
);

                /* Initialize the Slave I2C Driver */
                i2cSlaveObject  = DRV_I2C_Initialize(DRV_I2C_INDEX_1, (SYS_MODULE_INIT *)&drvI2C1InitData);

                /* Check for the System Status */
                if( SYS_STATUS_READY != DRV_I2C_Status( i2cObject ) )
                return 0;

                /* Open the Driver for I2C Master */
                drvI2CMasterHandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );
```

```
                if ( drvI2CMasterHandle != (DRV_HANDLE)NULL )
                {
                    /* event-handler set up receive callback from DRV_I2C_Tasks */
                    /* Event handler need to be set up only if needed */
                    DRV_I2C_BufferEventHandlerSet(drvI2CMasterHandle, I2CMasterOpStatusCb, i2cOpStatus );

                    /* Update the state to transfer data */
                    state = APP_STATE_DATA_PUT;
                }
                else
                {
                    state = APP_STATE_ERROR;
                }

                /* Open the I2C Driver for Slave on the same device */
                drvI2CSlaveHandle = DRV_I2C_Open( DRV_I2C_INDEX_1,DRV_IO_INTENT_WRITE );

                if ( drvI2CMasterHandle != (DRV_HANDLE)NULL )
                {
                    /* event-handler set up receive callback from DRV_I2C_Tasks */
                    /* Event handler need to be set up only if needed */
                    DRV_I2C_BufferEventHandlerSet(drvI2CMasterHandle, I2CMasterOpStatusCb, i2cOpStatus );

                    /* Update the state to transfer data */
                    state = APP_STATE_DATA_PUT;
                }
                else
                {
                    state = APP_STATE_ERROR;
                }


                break;
            }
            case APP_STATE_DATA_PUT:
            {
                /* I2C master writes data onto I2C bus */
                bufHandle_M1 = DRV_I2C_Transmit ( drvI2CMasterHandle , address, &myBuffer[], 5, NULL );

                /* Update the state to status check */
                state = APP_STATE_DATA_CHECK;
                break;
            }
            case APP_STATE_DATA_CHECK:
            {
                /* Check for the successful data transfer */
                if( DRV_I2C_BUFFER_EVENT_COMPLETE == DRV_I2C_TransferStatusGet
                                                    (drvI2CMasterHandle, bufHandle_M1) )
                {
                    /* Do this repeatedly */
                    state = APP_STATE_DATA_PUT;
                }
                break;
            }
            case APP_STATE_ERROR:
            {

                //include any error handling routines here

                break;
            }
            default:
            {
                break;
            }
        }
    }
```

```
/*****************************************************************************/
//  Function: I2CMasterOpStatusCb
//
//  Callback called in Master mode from the DRV_I2C_Tasks function. This
//  callback is invoked when the Master has to indicate to the application
//  that the BUFFER event is COMPLETE or there was an error in transmission.
//*****************************************************************************/

void I2CMasterOpStatusCb (    DRV_I2C_BUFFER_EVENT event,
                           DRV_I2C_BUFFER_HANDLE bufferHandle,
                           uintptr_t context)
{
    switch(event)
    {
        case DRV_I2C_BUFFER_EVENT_COMPLETE:
            //this indicates that the I2C transaction has completed
            //DRV_I2C_BUFFER_EVENT_COMPLETE can be handled in the callback
            //or by checking for this event using the API DRV_I2C_BufferStatus
            /* include any callback event handling code here if needed */
        break;
        case DRV_I2C_BUFFER_EVENT_ERROR:
            //this indicates that the I2C transaction has completed
            //and a STOP condition has been asserted on the bus.
            //However the slave has NACKED either the address or data
            //byte.
            /* include any callback event handling code here if needed */
        default:
        break;
    }

}

//*****************************************************************************/
//  Function: APP_SlaveDataforMaster
//
//  Callback function from DRV_I2C_Tasks when operating as a Slave. When an
//  address match is received by the Slave, this callback is executed and
//  the buffer event depends on the R/W bit. If R/W = 0, DRV_I2C_Receive is
//  called implying the Slave is going to read data send from the Master.
//  If R/W = 1, DRV_I2C_Transmit is called implying the Slave is going to send
//  data to the Master.
//*****************************************************************************/

void APP_SlaveDataforMaster(DRV_I2C_BUFFER_EVENT event, void * context)
{
    switch (event)
    {
        case DRV_I2C_BUFFER_SLAVE_READ_REQUESTED:
            deviceAddressPIC32 = PIC32_SLAVE_ADDRESS;

            bufHandle_S1 = DRV_I2C_Receive( drvI2CSlaveHandle,
                                            deviceAddressPIC32,
                                            &SlaveRxbuffer[0],
                                            NUMBER_OF_UNKNOWN_BYTES_TO_SLAVE,
                                            NULL );

            break;
        case DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED:
            deviceAddressPIC32 = PIC32_SLAVE_ADDRESS;

            bufHandle_S2 = DRV_I2C_Transmit ( drvI2CSlaveHandle,
                                              deviceAddressPIC32,
                                              &SlaveTxbuffer[0],
                                              NUMBER_OF_UNKNOWN_BYTES_TO_SLAVE,
                                              NULL );
            break;
        default:
            break;
    }
```

```
}

void __ISR(_I2C_2_VECTOR, ipl4AUTO) _IntHandlerDrvI2CInstance0(void)
{
   DRV_I2C_Tasks(i2cMasterObject);
}

void __ISR(_I2C1_SLAVE_VECTOR, ipl6AUTO) _IntHandlerDrvI2CSlaveInstance1(void)
{
    DRV_I2C_Tasks(i2cSlaveObject);
}
```

Configuring the Library

Macros

	Name	Description
	DRV_DYNAMIC_BUILD	Dynamic driver build, dynamic device instance parameters.
	DRV_I2C_CONFIG_BUILD_TYPE	Selects static or dynamic driver build configuration.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC	Enables the device driver to support basic transfer mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING	Enables the device driver to support blocking operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE	Enables the device driver to support operation in Exclusive mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER	Enables the device driver to support operation in Master mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING	Enables the device driver to support non-blocking during operations
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ	Enables the device driver to support read operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE	Enables the device driver to support operation in Slave mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE	Enables the device driver to support write operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ	Enables the device driver to support write followed by read.
	DRV_STATIC_BUILD	Static driver build, static device instance parameters.
	DRV_I2C_FORCED_WRITE	Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave
	I2C_STATIC_DRIVER_MODE	Selects the type of STATIC driver

Description

The configuration of the I2C Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the I2C Driver Library. Based on the selections made, the I2C Driver Library may support the selected features. These configuration settings will apply to all instances of the I2C Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_DYNAMIC_BUILD Macro

Dynamic driver build, dynamic device instance parameters.

File

drv_i2c_config_template.h

C

```
#define DRV_DYNAMIC_BUILD 1
```

Description

Dynamic Driver Build Configuration

This value, if used to identify the build type for a driver, will cause the driver to be built to dynamically, identify the instance of the peripheral at run-time using the parameter passed into its API routines.

DRV_I2C_CONFIG_BUILD_TYPE Macro

Selects static or dynamic driver build configuration.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_BUILD_TYPE DRV_DYNAMIC_BUILD
```

Description

I2C Driver Build Configuration Type

This definition selects if I2C device driver is to be used with static or dynamic build parameters. Must be equated to one of the following values:

- DRV_STATIC_BUILD - Build the driver using static accesses to the peripheral identified by the DRV_I2C_INSTANCE macro
- DRV_DYNAMIC_BUILD - Build the driver using dynamic accesses to the peripherals

Affects all the drv_i2c.h driver functions.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC Macro

Enables the device driver to support basic transfer mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC
```

Description

Support Basic Transfer Mode

This definition enables the device driver to support basic transfer mode.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_TransmitThenReceive

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING Macro

Enables the device driver to support blocking operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING
```

Description

Support Blocking Operations

This definition enables the device driver to support blocking operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open
- DRV_I2C_Close
- DRV_I2C_Receive
- DRV_I2C_Transmit
- DRV_I2C_TransmitThenReceive

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE Macro

Enables the device driver to support operation in Exclusive mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE
```

Description

Support Exclusive Mode

This definition enables the device driver to support operation in Exclusive mode.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER Macro

Enables the device driver to support operation in Master mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER
```

Description

Support Master Mode

This definition enables the device driver to support operation in Master mode.

Remarks

During the configuration phase, the driver selects a list of operation modes that can be supported. While initializing a hardware instance, the device driver will properly perform the initialization base on the selected modes.

The device driver can support multiple modes within a single build.

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING Macro

Enables the device driver to support non-blocking during operations

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING
```

Description

Support Non-Blocking Operations

This definition enables the device driver to support non-blocking operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open

- DRV_I2C_Close

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ Macro

Enables the device driver to support read operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ
```

Description

Support Read Mode

This definition enables the device driver to support read operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Receive

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE Macro

Enables the device driver to support operation in Slave mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE
```

Description

Support Slave Mode

This definition enables the device driver to support operation in Slave mode.

Remarks

During the configuration phase, the driver selects a list of operation modes that can be supported. While initializing a hardware instance, the device driver will properly perform the initialization base on the selected modes.

The device driver can support multiple modes within a single build.

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE Macro

Enables the device driver to support write operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE
```

Description

Support Write Mode

This definition enables the device driver to support write operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Transmit

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ Macro

Enables the device driver to support write followed by read.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ
```

Description

Support Write followed by a Read using Restart

This definition enables the device driver to support write followed by read without relinquishing control of the bus. Restart is issued instead of Stop at the end of write. Stop is issued after read operation.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_TransmitThenReceive

Refer to the description of each function in the corresponding help file for details.

DRV_STATIC_BUILD Macro

Static driver build, static device instance parameters.

File

drv_i2c_config_template.h

C

```
#define DRV_STATIC_BUILD 0
```

Description

Static Driver Build Configuration

This value, if used to identify the build type for a driver, will cause the driver to be built using a specific statically identified instance of the peripheral.

DRV_I2C_FORCED_WRITE Macro

Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_FORCED_WRITE true
```

Description

I2C driver objects configuration

When this option is checked, this will include Forced Write function. The Force Write function will send all data bytes to the slave irrespective of receiving ACK or NACK from slave. If writing data to the slave is invoked using DRV_I2C_Transfer, the transaction will be aborted if the Slave NACKs address or any data byte and a STOP condition will be send. This function is typically included for Slaves that require a special reset sequence.

Remarks

None

I2C_STATIC_DRIVER_MODE Macro

Selects the type of STATIC driver

File

drv_i2c_config_template.h

C

```
#define I2C_STATIC_DRIVER_MODE BUFFER_MODEL_STATIC
```

Description

I2C Static Driver type

This selects either the BYTE_MODEL_STATIC or BUFFER_MODEL_STATIC version of I2C driver. The BYTE_MODEL_STATIC version is equivalent to and is referred to as STATIC driver implementation in Harmony Versions 1.06.02 and below. This version of STATIC driver is not recommended for new design and will be deprecated in future release. The BUFFER_MODEL_STATIC supports transfer of buffers and is API compatible with the DYNAMIC implementation of I2C.

Building the Library

This section lists the files that are available in the I2C Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/i2c`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_i2c.h`	This file provides the interface definitions of the I2C driver.
`/drv_i2c_bb.h`	This file provides interface definitions that are transparent to the user when the I2C Driver is used in Bit-bang mode.
`/src/drv_i2c_local.h`	This file provides definitions of the data types that are used in the driver object.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_i2c.c`	This file contains the core implementation of the I2C driver.
`/src/dynamic/drv_i2c_bb.c`	This file implements the I2C Driver in Bit-bang mode.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The I2C Driver Library depends on the following modules:

- Clock System Service Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_I2C_Deinitialize	Deinitializes the index instance of the I2C module. **Implementation:** Static/Dynamic

	DRV_I2C_Initialize	Initializes hardware and data for the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Tasks	Maintains the State Machine of the I2C driver and performs all the protocol level actions. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_I2C_Close	Closes an opened instance of an I2C module driver. **Implementation:** Dynamic
	DRV_I2C_Open	Opens the specified instance of the I2C driver for use and provides an "open-instance" handle. **Implementation:** Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_I2C_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2C_BytesTransferred	Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.
	DRV_I2C_Receive	This function reads data written from either Master or Slave. **Implementation:** Dynamic
	DRV_I2C_Transmit	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitThenReceive	This function writes data to Slave, inserts restart and requests read from slave. **Implementation:** Dynamic
	DRV_I2C_TransmitForced	This function writes data to Master or Slave. **Implementation:** Dynamic

d) Status Functions

	Name	Description
	DRV_I2C_TransferStatusGet	Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver. **Implementation:** Dynamic
	DRV_I2C_Status	Provides the current status of the index instance of the I2C module. **Implementation:** Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_I2C_QueueFlush	The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.
	DRV_I2C_SlaveCallbackSet	Allows a client to identify a Slave Callback function for the driver to call back when drivers needs to initiate a read or write operation.

f) Data Types and Constants

	Name	Description
	DRV_I2C_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_I2C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2C_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2C_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_I2C_BB_H	This is macro DRV_I2C_BB_H.

Description

This section describes the Application Programming Interface (API) functions of the I2C Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_I2C_Deinitialize Function

Deinitializes the index instance of the I2C module.

Implementation: Static/Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the index instance of the I2C module, disabling its operation (and any hardware for driver modules). It deinitializes only the specified module instance. It also resets all the internal data structures and fields for the specified instance to the default settings.

Remarks

If the module instance has to be used again, DRV_I2C_Initialize should be called again to initialize the module instance structures.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will NEVER block for hardware I2C access. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_I2C_Status operation. The driver client must always use DRV_I2C_Status to find out when the module is in the ready state.

Preconditions

The DRV_I2C_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS   i2c_status;

DRV_I2C_Deinitialize(I2C_ID_1);

i2c_status = DRV_I2C_Status(I2C_ID_1);
if (SYS_STATUS_BUSY == i2c_status)
{
    // Do something else and check back later
}
else if (SYS_STATUS_ERROR >= i2c_status)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be deinitialized

Function

void DRV_I2C_Deinitialize (SYS_MODULE_OBJ object)

DRV_I2C_Initialize Function

Initializes hardware and data for the index instance of the I2C module.

Implementation: Static/Dynamic

File

drv_i2c.h

C

```
SYS_MODULE_OBJ DRV_I2C_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function initializes hardware for the index instance of the I2C module, using the hardware initialization given data. It also initializes any internal driver data structures making the driver ready to be opened.

Remarks

This function must be called before any other I2C function is called.

This function should only be called once during system initialization unless DRV_I2C_Deinitialize is first called to deinitialize the device instance before reinitializing it.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will NEVER block for hardware I2C access. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_I2C_Status operation. The driver client must always use DRV_I2C_Status to find out when the module is in the ready state.

Whenever a call to DRV_I2C_Initialize is made with a SYS_MODULE_INIT* data == 0 the following default configuration will be used. Adjust this configuration at build time as needed.

Preconditions

None.

Example

```
DRV_I2C_INIT        i2c_init_data;
SYS_MODULE_OBJ      objectHandle;

i2c_init_data.i2cId = DRV_I2C_PERIPHERAL_ID_IDX0,
i2c_init_data.i2cMode = DRV_I2C_MODE_MASTER,

OR

i2c_init_data.i2cMode = DRV_I2C_MODE_SLAVE,
//Master mode parameters
i2c_init_data.baudRate = 100000,
i2c_init_data.busspeed = DRV_I2C_SLEW_RATE_CONTROL_IDX0,
i2c_init_data.buslevel = DRV_I2C_SMBus_SPECIFICATION_IDX0,

//Slave mode parameters
i2c_init_data.addWidth = DRV_I2C_7BIT_SLAVE,
i2c_init_data.reservedaddenable  = false,
i2c_init_data.generalcalladdress = false,
i2c_init_data.slaveaddvalue = 0x0060,

//interrupt sources
i2c_init_data.mstrInterruptSource = INT_SOURCE_I2C_2_MASTER,
i2c_init_data.slaveInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.errInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.queueSize = 1,

//callback for Master (Master mode can use callbacks if needed)
i2c_init_data.operationStarting = NULL,
// Slave mode callbacks needed
i2c_init_data.operationStarting = APP_I2CSlaveFunction(),
i2c_init_data.operationEnded = NULL

objectHandle = DRV_I2C_Initialize(DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData)
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be initialized

data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and the default initialization is to be used.

Function

void DRV_I2C_Initialize (const I2C_MODULE_ID index,
const SYS_MODULE_INIT *const data)

DRV_I2C_Tasks Function

Maintains the State Machine of the I2C driver and performs all the protocol level actions.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Tasks(SYS_MODULE_OBJ object);
```

Description

This functions maintains the internal state machine of the I2C driver. This function acts as the I2C Master or Slave ISR. When used in polling mode, this function needs to be called repeatedly to achieve I2C data transfer. This function implements all the protocol level details like setting the START condition, sending the address with with R/W request, writing data to the SFR, checking for acknowledge and setting the STOP condition.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance.

Example

SYS_MODULE_OBJ object;
while (true) { DRV_I2C_Tasks (object);

Function

void DRV_I2C_Tasks (SYS_MODULE_OBJ object)

b) Client Setup Functions

DRV_I2C_Close Function

Closes an opened instance of an I2C module driver.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of an I2C module driver, making the specified handle invalid.

Remarks

After calling This function, the handle passed into drvHandle must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_I2C_Open before the caller may use the driver again.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
myI2CHandle = DRV_I2C_Open(I2C_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

// Perform data transfer operations

DRV_I2C_Close(myI2CHandle);
```

Parameters

Parameters	Description
drvHandle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2C_Close (const DRV_HANDLE drvHandle)

DRV_I2C_Open Function

Opens the specified instance of the I2C driver for use and provides an "open-instance" handle.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_HANDLE DRV_I2C_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a value identifying both the caller and the module instance). If an error occurs, the returned value is DRV_HANDLE_INVALID.

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_I2C_INSTANCES_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This function opens the specified instance of the I2C module for use and provides a handle that is required to use the remaining driver routines.

This function opens a specified instance of the I2C module driver for use by any client module and provides an "open-instance" handle that must be provided to any of the other I2C driver operations to identify the caller and the instance of the I2C driver/hardware module.

Remarks

The handle returned is valid until the DRV_I2C_Close routine is called.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. Regarding the hardware I2C access the operation will behave as instructed by the DRV_IO_INTENT parameter.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

Example

```
DRV_HANDLE              i2c_handle;

i2c_handle = DRV_I2C_Open(I2C_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == i2c_handle)
{
    // Handle open error
}

// Close the device when it is no longer needed.
DRV_I2C_Close(i2c_handle);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be opened.
intent	Flags parameter identifying the intended usage and behavior of the driver. Multiple flags may be ORed together to specify the intended usage of the device. See the DRV_IO_INTENT definition.

Function

DRV_HANDLE DRV_I2C_Open (const I2C_MODULE_ID index,
const DRV_IO_INTENT intent)

c) Data Transfer Functions

DRV_I2C_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_BufferEventHandlerSet(const DRV_HANDLE handle, const DRV_I2C_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when a queued buffer transfer has finished. When a client calls either the DRV_I2C_Receive, DRV_I2C_Transmit or DRV_I2C_TransmiThenReceive function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any transmision or reception operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

When in Master mode, a callback event is registered to let the application know that the buffer has been transmitted. DRV_I2C_BUFFER_EVENT_COMPLETE is set when the buffer has been transmitted without any errors. DRV_I2C_BUFFER_EVENT_ERROR is set when buffer transmission or reception has been aborted.

When in Slave mode, since the Master controls when a transmit or receive operation is terminated, a callback is registered every time a byte is written or read from the slave.

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C driver instance.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
#define MY_BUFFER_SIZE  10

// function prototype of Event Handler Function
void APP_I2CBufferEventFunction ( DRV_I2C_BUFFER_EVENT event,
                                  DRV_I2C_BUFFER_HANDLE bufferHandle,
                                  uintptr_t context );

//Returned from DRV_I2C_Open
DRV_HANDLE drvI2Chandle;

// myAppObj is an application specific state data object.
```

```
DRV_I2C_BUFFER_EVENT operationStatus;

uint8_t appBuffer[MY_BUFFER_SIZE];

DRV_I2C_BUFFER_HANDLE drvI2CRDBUFHandle

// Opens an instance of I2C driver
drvI2Chandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

// Client registers an event handler with driver. This is done once.
DRV_I2C_BufferEventHandlerSet( drvI2Chandle,
                               APP_I2CBufferEventFunction,
                               operationStatus );

drvI2CRDBUFHandle = DRV_I2C_Receive (   drvI2CHandle,
                                        slaveaddress
                                        &appBuffer[],
                                        MY_BUFFER_SIZE,
                                        NULL );

if(NULL == drvI2CRDBUFHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2CBufferEventFunction( DRV_I2C_BUFFER_EVENT event,
                                 DRV_I2C_BUFFER_HANDLE handle,
                                 uintptr_t context)
{

    switch(event)
    {
        case DRV_I2C_BUFFER_EVENT_COMPLETE:
            //perform appropriate action
        break;

        case DRV_I2C_BUFFER_EVENT_ERROR:
            // Error handling here.
        break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_I2C_BufferEventHandlerSet (

const DRV_HANDLE handle,

const DRV_I2C_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t context)

DRV_I2C_BytesTransferred Function

Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.

File

drv_i2c.h

C

```
uint32_t DRV_I2C_BytesTransferred(DRV_HANDLE handle, DRV_I2C_BUFFER_HANDLE bufferHandle);
```

Returns

The number of bytes transferred in a particular I2C transaction.
`numOfBytes = DRV_I2C_BytesTransferred (drvI2CHandle_Master,drvBufferHandle);`

Description

This returns the transmitter and receiver transfer status.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
bufferHandle	A valid buffer handle obtained when calling Transmit/Receive/TransmitThenReceive/TransmitForced or BufferAddRead/BufferAddWrite/BufferAddReadWrite function

Function

uint32_t DRV_I2C_BytesTransferred (DRV_I2C_BUFFER_HANDLE bufferHandle)

DRV_I2C_Receive Function

This function reads data written from either Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_Receive(DRV_HANDLE handle, uint16_t address, void * buffer, size_t size, void
* callbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to read data send by Slave. The Slave calls this function to read data send by Master. In case of Master, a START condition is initiated on the I2C bus.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If the slave NACKs the address byte, then further read is not attempted. Master asserts STOP condition and DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If all the requisite number of bytes have been read then DRV_I2C_BUFFER_EVENT_COMPLETE is set as the buffer status.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CRDBUFHandle = DRV_I2C_Receive( drvI2CHandle,
                                     deviceaddress,
                                     &rxbuffer[0],
                                     num_of_bytes,
                                     NULL );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave shifted so that bits 7 - 1 are address
bits A6	A0. This value is Ignored in slave mode.
buffer	This buffer holds data is received
size	The number of bytes that the Master expects to read from Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the READ operation is terminated.
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_Receive (DRV_HANDLE handle,

uint16_t slaveaddress,

void *rxBuffer,

size_t size,

void * callbackContext)

DRV_I2C_Transmit Function

This function writes data to Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_Transmit(DRV_HANDLE handle, uint16_t slaveaddress, void * buffer, size_t size, void * context);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to write data to Slave. The Slave calls this function to write data to Master.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If the slave NACKs the address byte or any data bytes, then further write is not attempted. Master asserts STOP condition and DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If all the requisite number of bytes have been transmitted to the Slave, then DRV_I2C_BUFFER_EVENT_COMPLETE is set as the buffer status.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CWRBUFHandle = DRV_I2C_Transmit( drvI2CHandle,
                                      deviceaddress,
                                      &txBuffer[0],
                                      num_of_bytes,
                                      NULL);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave shifted so that bits 7 - 1 are address
bits A6	A0. This value is Ignored in slave mode.
buffer	Contains data to be transferred

size	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_Transmit(DRV_HANDLE handle,
uint16_t slaveaddress,
void *txBuffer,
size_t size,
void *context);

DRV_I2C_TransmitThenReceive Function

This function writes data to Slave, inserts restart and requests read from slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitThenReceive(DRV_HANDLE handle, uint16_t address, void * writeBuffer,
size_t writeSize, void * readBuffer, size_t readSize, void * callbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to send a register address value to the slave and then queries the slave with a read request to read the contents indexed by the register location. The Master sends a restart condition after the initial write before sending the device address with R/W = 1. The restart condition prevents the Master from relinquishing the control of the bus. The slave should not use this function.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If there is any error condition during transmission then further transmission or reception is not attempted and STOP condition is asserted on the bus. In case of error condition, DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If the I2C bus transaction is completed as requested then the buffer status, is set as DRV_I2C_BUFFER_EVENT_COMPLETE.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CRDBUFHandle = DRV_I2C_TransmiThenReceive( appData.drvI2CHandle,
                                                deviceaddress,
                                                &drvI2CTXbuffer[0],
                                                registerbytesize,
                                                rxbuffer,
                                                num_of_bytes,
                                                NULL );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave shifted so that bits 7 - 1 are address
bits A6	A0. This value is Ignored in slave mode.
writeBuffer	Contains data to be transferred
writeSize	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
readBuffer	This buffer holds data that is send back from slave after read operation.

readSize	The number of bytes the Master expects to be read from the slave
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmiThenReceive (DRV_HANDLE handle,
uint16_t deviceaddress,
void *txBuffer,
size_t writeSize,
void *rxBuffer,
size_t readSize,
void *context)

DRV_I2C_TransmitForced Function

This function writes data to Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitForced(DRV_HANDLE handle, uint16_t deviceaddress, void* txBuffer,
size_t txbuflen, DRV_I2C_BUS_ERROR_EVENT eventFlag, void * calbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to transmit the entire buffer to the slave even if the slave ACKs or NACKs the address or any of the data bytes. This is typically used for slaves that have to initiate a reset sequence by sending a dummy I2C transaction. Since the slave is still in reset, any or all the bytes can be NACKed. In the normal operation of the driver if the address or data byte is NACKed, then the transmission is aborted and a STOP condition is asserted on the bus.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULl value is returned. Once all the bytes are transferred the buffer status is set as then DRV_I2C_BUFFER_EVENT_COMPLETE .

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CWRBUFHandle = DRV_I2C_TransmitForced ( handle,
                                             deviceaddress,
                                             &txBuffer[0],
                                             txbuflen,
                                             NULL,
                                             NULL)
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave shifted so that bits 7 - 1 are address
bits A6	A0. This value is Ignored in slave mode.
buffer	Contains data to be transferred
size	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
eventFlag	This field is left for future implementation

callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitForced (DRV_HANDLE handle,
uint16_t deviceaddress,
uint8_t* txBuffer,
uint16_t txbuflen,
DRV_I2C_BUS_ERROR_EVENT eventFlag,
void * calbackContext)

d) Status Functions

DRV_I2C_TransferStatusGet Function

Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_EVENT DRV_I2C_TransferStatusGet(DRV_HANDLE handle, DRV_I2C_BUFFER_HANDLE bufferHandle);
```

Returns

A DRV_I2C_TRANSFER_STATUS value describing the current status of the transfer.

Description

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event. If the event is DRV_I2C_BUFFER_EVENT_COMPLETE, it means that the data was transferred successfully. If the event is DRV_I2C_BUFFER_EVENT_ERROR, it means that the data was not transferred successfully.

Remarks

The handle that is passed into the function, drvI2CBUFHandle is obtained by calling one of the data transfer functions. The drvI2CBUFHandle should be a valid handle and not a NULL value. The DRV_I2C_BufferStatus can be called to check the progress of the data transfer operation. If the buffer is transferred without any error, then DRV_I2C_BUFFER_EVENT_COMPLETE is returned. If an error condition is present, then DRV_I2C_BUFFER_EVENT_ERROR is returned.

Example

```
if(DRV_I2C_BUFFER_EVENT_COMPLETE == DRV_I2C_TransferStatusGet ( handle,
                                                                bufferHandle ))
{
        //perform action
        return true;
}
else
{
        //perform action
        return false;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
bufferHandle	A valid buffer handle obtained when calling Transmit/Receive/TransmitThenReceive/TransmitForced or BufferAddRead/BufferAddWrite/BufferAddReadWrite function

Function

DRV_I2C_BUFFER_EVENT DRV_I2C_TransferStatusGet (DRV_HANDLE handle,
DRV_I2C_BUFFER_HANDLE bufferHandle)

DRV_I2C_Status Function

Provides the current status of the index instance of the I2C module.

Implementation: Dynamic

File

drv_i2c.h

C

```
SYS_STATUS DRV_I2C_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified I2Cmodule has completed.
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified I2C module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified I2C module is in an error state

Description

This function provides the current status of the index instance of the I2C module.

Remarks

The DRV_I2C_Status operation can be used to determine when any of the I2C module level operations has completed. The value returned by the DRV_I2C_Status routine hast to be checked after calling any of the I2C module operations to find out when they have completed.

If the DRV_I2C_Status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the DRV_I2C_Status operation return SYS_STATUS_READY, any previous operations have completed.

The DRV_I2C_Status function will NEVER block.

If the DRV_I2C_Status operation returns an error value, the error may be cleared by calling the DRV_I2C_Initialize operation. If that fails, the DRV_I2C_Deinitialize operation will need to be called, followed by the DRV_I2C_Initialize operation to return to normal operations.

Preconditions

The DRV_I2C_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;
SYS_STATUS          i2c_status;

i2c_status = DRV_I2C_Status(object);
if (SYS_STATUS_BUSY == i2c_status)
{
    // Do something else and check back later
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to get status for.

Function

SYS_STATUS DRV_I2C_Status (SYS_MODULE_OBJ object)

e) Miscellaneous Functions

DRV_I2C_QueueFlush Function

The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.

File

drv_i2c.h

C

```
void DRV_I2C_QueueFlush(DRV_HANDLE handle);
```

Returns

None

```
    //Opens an instance of I2C driver
 drvI2Chandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

 DRV_I2C_QueueFlush ( drvI2CHandle );
```

Description

The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2C_QueueFlush (DRV_HANDLE handle)

DRV_I2C_SlaveCallbackSet Function

Allows a client to identify a Slave Callback function for the driver to call back when drivers needs to initiate a read or write operation.

File

drv_i2c.h

C

```
void DRV_I2C_SlaveCallbackSet(const DRV_HANDLE handle, const DRV_I2C_CallBack callback, const uintptr_t
context);
```

Returns

None

```
 #define APP_I2C_BUFFER_SIZE      10

 void APP_I2C_SlaveTransferCallback( DRV_I2C_BUFFER_EVENT event, void * context );

 uint8_t slaveBuffer[ APP_I2C_BUFFER_SIZE ];

 DRV_HANDLE drvI2CHandle;

 void APP_I2C_SlaveTransferCallback( DRV_I2C_BUFFER_EVENT event, void * context )
 {
     switch (event)
     {
         case DRV_I2C_BUFFER_SLAVE_READ_REQUESTED:
         {
             appData.bufferReceive = DRV_I2C_Receive( drvI2CHandle,
                                                      0,
                                                      &slaveBuffer[0],
                                                      APP_I2C_BUFFER_SIZE,
                                                      NULL );
             break;
         }

         case DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED:
         {
             appData.bufferTransmit = DRV_I2C_Transmit ( drvI2CHandle,
                                                         0,
                                                         &slaveBuffer[0],
                                                         APP_I2C_BUFFER_SIZE,
                                                         NULL );
             break;
         }
```

```
            default:
            {
                break;
            }
        }

        return;
    }

    // Opens an instance of I2C driver
    drvI2Chandle = DRV_I2C_Open( DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE );

    // Set the operation callback
    DRV_I2C_SlaveCallbackSet( drvI2Chandle, APP_I2C_SlaveTransferCallback, 0);
```

Description

This function allows a client to identify a Slave Callback function for the driver to call back when drivers needs to initiate a read or write operation. When the I2C Slave driver receives a read or write event from master the callback is called to initiate the user defined action. The callback should be set before the master requests for read or write operations. The event handler once set, persists until the client closes the driver or sets another event handler.

In Slave mode, a callback event is registered to let the application know that master has requested for either read or write operation. DRV_I2C_BUFFER_SLAVE_READ_REQUESTED is set when the master sends data and wants slave to read. DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED is set when the master tries to read data from slave and wants slave to send the data.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
callback	pointer to the callback function.
context	The value of parameter will be passed back to the client unchanged, when the callback function is called. It can be used to identify any client specific data

Function

void DRV_I2C_SlaveCallbackSet (const DRV_HANDLE handle,

const DRV_I2C_CallBack callback,

const uintptr_t context)

f) Data Types and Constants

DRV_I2C_BUFFER_QUEUE_SUPPORT Macro

Specifies if the Buffer Queue support should be enabled.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_BUFFER_QUEUE_SUPPORT false
```

Description

I2C Driver Buffer Queue Support

This macro defines if Buffer Queue support should be enabled. Setting this macro to true will enable buffer queue support and all buffer related driver function.

Remarks

None

DRV_I2C_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_INSTANCES_NUMBER 5
```

Description

I2C driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of I2C modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None

DRV_I2C_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_INTERRUPT_MODE true
```

Description

I2C Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values it can take are

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, then Interrupt Service Routine for the interrupt should be defined in the application. The DRV_I2C_Tasks() routine should be called in the ISR.

Remarks

None

DRV_I2C_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_QUEUE_DEPTH_COMBINED 7
```

Description

I2C Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_I2C_Transmit() function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_I2C_Receive() function.

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all I2C driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the

minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

In the current implementation of I2C driver, queueing of Buffers is not supported. This will be added in a future release.

Remarks

None

DRV_I2C_BB_H Macro

File

drv_i2c_bb.h

C

```
#define DRV_I2C_BB_H
```

Description

This is macro DRV_I2C_BB_H.

Files

Files

Name	Description
drv_i2c.h	I2C module driver interface header.
drv_i2c_bb.h	Contains prototypes for the I2C functions
drv_i2c_config_template.h	I2C device driver configuration file.

Description

drv_i2c.h

I2C module driver interface header.

Functions

	Name	Description
	DRV_I2C_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2C_BytesTransferred	Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.
	DRV_I2C_Close	Closes an opened instance of an I2C module driver. **Implementation:** Dynamic
	DRV_I2C_Deinitialize	Deinitializes the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Initialize	Initializes hardware and data for the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Open	Opens the specified instance of the I2C driver for use and provides an "open-instance" handle. **Implementation:** Dynamic
	DRV_I2C_QueueFlush	The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.
	DRV_I2C_Receive	This function reads data written from either Master or Slave. **Implementation:** Dynamic
	DRV_I2C_SlaveCallbackSet	Allows a client to identify a Slave Callback function for the driver to call back when drivers needs to initiate a read or write operation.
	DRV_I2C_Status	Provides the current status of the index instance of the I2C module. **Implementation:** Dynamic
	DRV_I2C_Tasks	Maintains the State Machine of the I2C driver and performs all the protocol level actions. **Implementation:** Dynamic

	DRV_I2C_TransferStatusGet	Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver. **Implementation:** Dynamic
	DRV_I2C_Transmit	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitForced	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitThenReceive	This function writes data to Slave, inserts restart and requests read from slave. **Implementation:** Dynamic

Description

I2C Device Driver Interface Header File

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the I2C module driver.

File Name

drv_i2c.h

Company

Microchip Technology Inc.

drv_i2c_bb.h

Contains prototypes for the I2C functions

Macros

	Name	Description
	DRV_I2C_BB_H	This is macro DRV_I2C_BB_H.

Description

I2C Bit Bang Functions Header File

File Name

drv_i2c_bb.h

Company

Microchip Technology Inc.

drv_i2c_config_template.h

I2C device driver configuration file.

Macros

	Name	Description
	DRV_DYNAMIC_BUILD	Dynamic driver build, dynamic device instance parameters.
	DRV_I2C_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_I2C_CONFIG_BUILD_TYPE	Selects static or dynamic driver build configuration.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC	Enables the device driver to support basic transfer mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING	Enables the device driver to support blocking operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE	Enables the device driver to support operation in Exclusive mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER	Enables the device driver to support operation in Master mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING	Enables the device driver to support non-blocking during operations
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ	Enables the device driver to support read operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE	Enables the device driver to support operation in Slave mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE	Enables the device driver to support write operations.

	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ	Enables the device driver to support write followed by read.
	DRV_I2C_FORCED_WRITE	Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave
	DRV_I2C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2C_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2C_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_STATIC_BUILD	Static driver build, static device instance parameters.
	I2C_STATIC_DRIVER_MODE	Selects the type of STATIC driver

Description

I2C Device Driver Configuration

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_i2c_config.h

Company

Microchip Technology Inc.

I2S Driver Library Help

This section describes the I2S Driver Library.

Introduction

This library provides an interface to manage the Audio Protocol Interface Modes of the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers.

Description

The SPI module can be interfaced to most available codec devices to provide microcontroller-based audio solutions. The SPI module provides support to the audio protocol functionality via four standard I/O pins. The four pins that make up the audio protocol interface modes are:

- SDIx: Serial Data Input for receiving sample digital audio data (ADCDAT)
- SDOx: Serial Data Output for transmitting digital audio data (DACDAT)
- SCKx: Serial Clock, also known as bit clock (BCLK)
- /SSx: Left/Right Channel Clock (LRCK)

BCLK provides the clock required to drive the data out or into the module, while LRCK provides the synchronization of the frame based on the protocol mode selected.

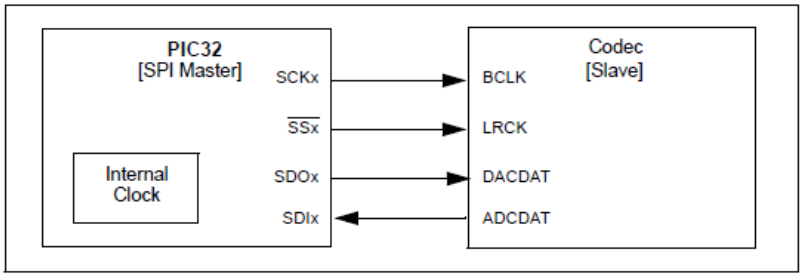
In Master mode, the module generates both the BCLK on the SCKx pin and the LRCK on the /SSx pin. In certain devices, while in Slave mode, the module receives these two clocks from its I2S partner, which is operating in Master mode.

When configured in Master mode, the leading edge of SCK and the LRCK are driven out within one SCK period of starting the audio protocol. Serial data is shifted in or out with timings determined by the protocol mode set.

In Slave mode, the peripheral drives zeros out SDO, but does not transmit the contents of the transmit FIFO until it sees the leading edge of the LRCK, after which time it starts receiving data.

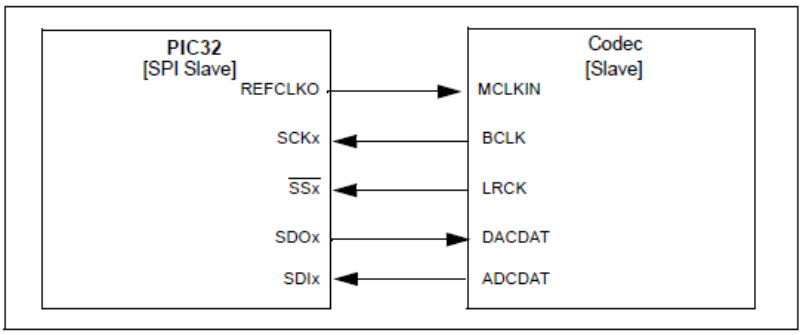
Master Mode

Master Generating its Own Clock – Output BCLK and LRCK



Slave Mode

Codec Device as Master Derives MCLK from PIC32 Reference Clock Out



Audio Protocol Modes

The SPI module supports four audio protocol modes and can be operated in any one of these modes:

- I2S mode
- Left-Justified mode
- Right-Justified mode
- PCM/DSP mode

Using the Library

This topic describes the basic architecture of the I2S Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_i2s.h`

The interface to the I2S Driver Library is defined in the `drv_i2s.h` header file. Any C language source (`.c`) file that uses the I2S Driver Library should include `drv_i2s.h`.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The SPI Peripheral Library provides the low-level abstraction of the SPI module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the I2S Driver Library interface.

Description

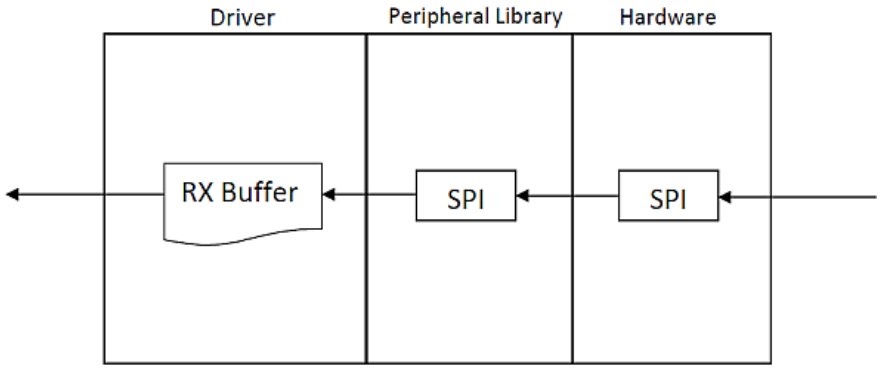
I2S Software Abstraction Block Diagram

Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not. The buffer depth varies across part families. The SPI Peripheral Library provides the ability to access these buffers. The I2S Driver Library abstracts out these differences and provides a unified model for audio data transfer across different types of SPI modules.

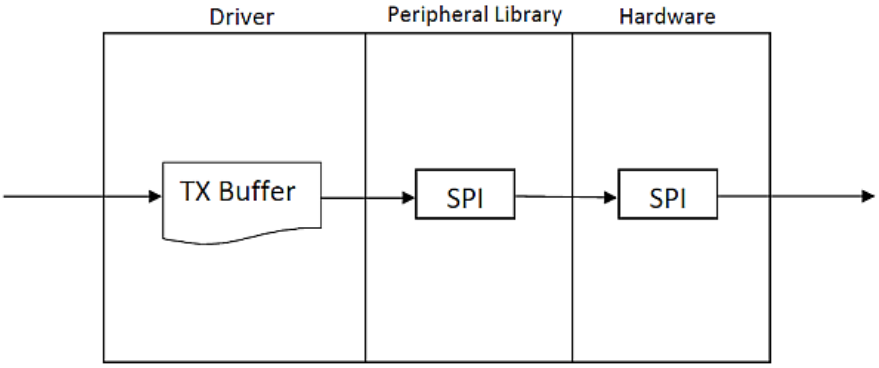
Both the transmitter and receiver provide a buffer in the driver, which transmits and receives data to/from the hardware. The I2S Driver Library provides a set of interfaces to perform the read and the write.

The following diagrams illustrate the model used by the I2S Driver Library for the transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The I2S driver library provides an API interface to transfer/receive digital audio data using supported Audio protocols. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2S Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Data Transfer Functions	Provides data transfer functions.
Miscellaneous Functions	Provides driver miscellaneous functions such as baud rate setting, get error functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the I2S Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

System Access

This section provides information on system access.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the I2S module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_I2S_INIT or by using Initialization Overrides) that are supported by the specific I2S device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., SPI_ID_2)
- Defining the respective interrupt sources for TX, RX, DMA TX Channel, DMA RX Channel and Error Interrupt

The DRV_I2S_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_I2S_Deinitialize, DRV_I2S_Status, DRV_I2S_Tasks, and DRV_I2S_TasksError.

Notes:

1. The system initialization setting only effect the instance of the peripheral that is being initialized.
2. Configuration of the dynamic driver for DMA mode(uses DMA channel for data transfer) or Non DMA mode can be performed by appropriately setting the 'dmaChannelTransmit' and 'dmaChannelReceive' variables of the DRV_I2S_INIT structure. For example the TX will be in DMA mode when 'dmaChannelTransmit' is initialized to a valid supported channel number from the enum DMA_CHANNEL. TX will be in Non DMA mode when 'dmaChannelTransmit' is initialized to 'DMA_CHANNEL_NONE'.

Example:

```
    DRV_I2S_INIT                  init;
    SYS_MODULE_OBJ              objectHandle;

    init.moduleInit.value           = SYS_MODULE_POWER_RUN_FULL;
    init.spiID                      = SPI_ID_1;
    init.usageMode                  = DRV_I2S_MODE_MASTER;
    init.baudClock                  = SPI_BAUD_RATE_MCLK_CLOCK;
    init.baud                       = 48000;
    init.clockMode                  = DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL;
    init.audioCommWidth             = SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL;
    init.audioTransmitMode          = SPI_AUDIO_TRANSMIT_STEREO;
    init.inputSamplePhase           = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
    init.protocolMode               = DRV_I2S_AUDIO_I2S;
    init.txInterruptSource          = INT_SOURCE_SPI_1_TRANSMIT;
    init.rxInterruptSource          = INT_SOURCE_SPI_1_RECEIVE;
    init.errorInterruptSource       = INT_SOURCE_SPI_1_ERROR;
    init.queueSizeTransmit          = 3;
    init.queueSizeReceive           = 2;
    init.dmaChannelTransmit         = DMA_CHANNEL_NONE;
    init.dmaChannelReceive          = DMA_CHANNEL_NONE;

  objectHandle = DRV_I2S_Initialize(DRV_I2S_INDEX_1, (SYS_MODULE_INIT*)init);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Task Routine

In a polled environment, the system will call DRV_I2S_Tasks and DRV_I2S_TasksError from the System Task Service. In an interrupt-based implementation, DRV_I2S_Tasks and DRV_I2S_TasksError will be called from the Interrupt Service Routine of the I2S. When a DMA channel is used for transmission/reception DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.

Client Access

This section provides information on general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_I2S_Open function. This provides the settings required to open the I2S instance for operation. If the driver is deinitialized using the function DRV_I2S_Deinitialize, the application must call the DRV_I2S_Open function again to set up the instance of the I2S.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Example:

```
DRV_HANDLE handle;
handle = DRV_I2S_Open(DRV_I2S_INDEX_0, (DRV_IO_INTENT_WRITE | DRV_IO_INTENT_NONBLOCKING));
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Client Operations - Buffered

This section provides information on buffered client operations.

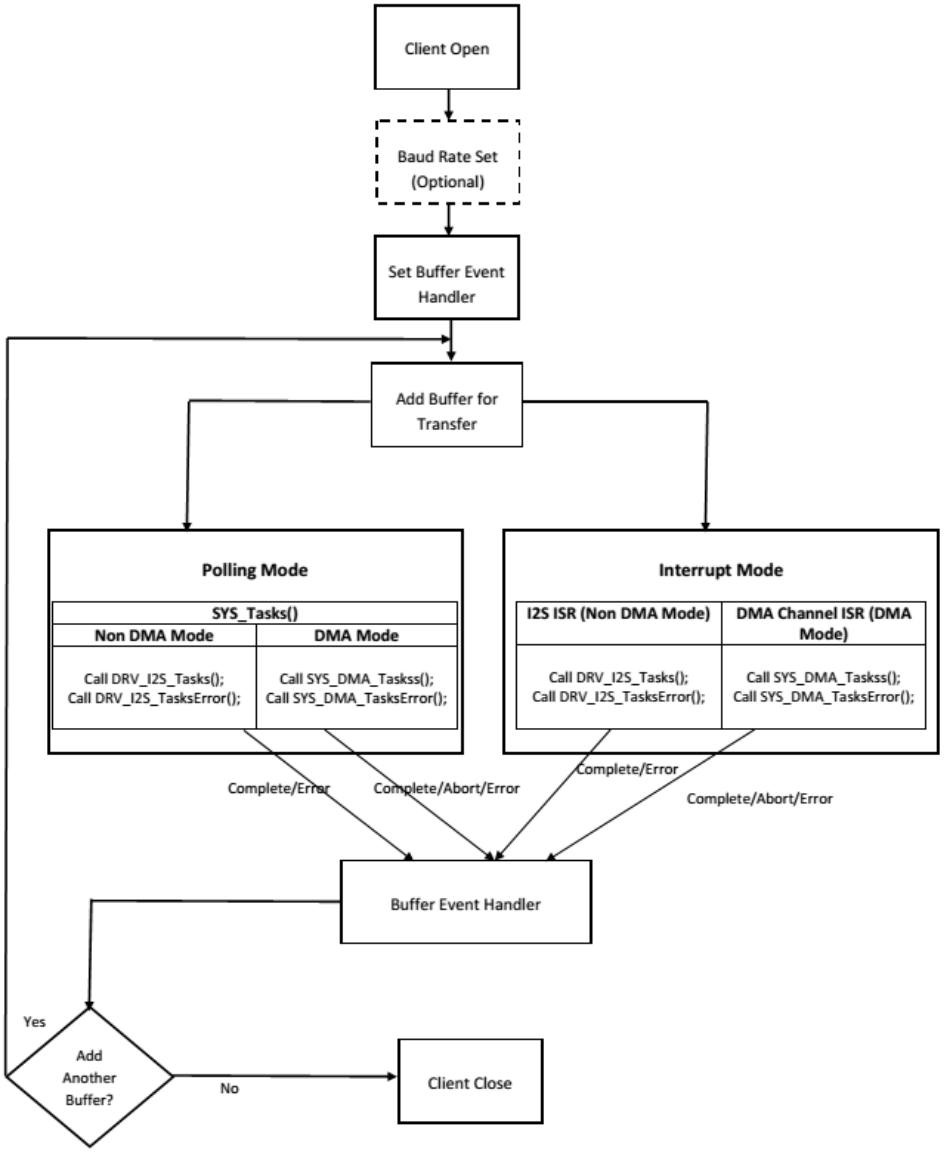
Description

Client Operations - Buffered

Client buffered operations provide a the typical audio interface. The functions DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite, and DRV_I2S_BufferAddWriteRead are the buffered data operation functions. The buffered functions schedules non-blocking operations. The function adds the request to the hardware instance queues and returns a buffer handle. The requesting client also registers a callback event with the driver. The driver notifies the client with DRV_I2S_BUFFER_EVENT_COMPLETE, DRV_I2S_BUFFER_EVENT_ERROR or DRV_I2S_BUFFER_EVENT_ABORT events.

The buffer add requests are processed under DRV_I2S_Tasks, DRV_I2S_TasksError functions. These functions are called from the I2S channel ISR in interrupt mode or from SYS_Tasks routine in Polled mode. When a DMA channel is used for transmission/reception DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.

The following diagram illustrates the buffered data operations



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. If DMA mode is desired, the DMA should be initialized by calling SYS_DMA_Initialize.
3. The necessary ports setup and remapping must be done for I2S lines: ADCDAT, DACDAT, BCLK, LRCK and MCLK (if required).
4. The driver object should have been initialized by calling DRV_I2S_Initialize. If DMA mode is desired, related attributes in the init structure must be set.
5. Open the driver using DRV_I2S_Open with the necessary ioIntent to get a client handle.
6. The necessary BCLK, LRCK, and MCLK should be set up so as to generate the required media bit rate.
7. The necessary Baud rate value should be set up by calling DRV_I2S_BaudrateSet.
8. The Register and event handler for the client handle should be set up by calling DRV_I2S_BufferEventHandlerSet.
9. Add a buffer to initiate the data transfer by calling DRV_I2S_BufferAddWrite/DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWriteRead.
10. Based on polling or interrupt mode service the data processing should be set up by calling DRV_I2S_Tasks, DRV_I2S_TasksError from system tasks or I^2S ISR. When a DMA channel is used for transmission/reception system calls SYS_DMA_Tasks(), SYS_DMA_TasksError() from the system tasks or DMA channel ISR, DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.
11. Repeat step 9 through step 10 to handle multiple buffer transmission and reception.

12. When the client is done it can use DRV_I2S_Close to close the client handle.

Example 1:

```
// The following is an example for a Polled mode buffered transmit

#define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
#define BUFFER_SIZE 1000
// I2S initialization structure.
// This should be populated with necessary settings.
// attributes dmaChannelTransmit/dmaChannelReceive
// and dmaInterruptTransmitSource/dmaInterruptReceiveSource
// must be set if DMA mode of operation is desired.
DRV_I2S_INIT i2sInit;
SYS_MODULE_OBJ sysObj;      //I2S module object
DRV_HANDLE handle;          //Client handle
uint32_t i2sClock;          //BCLK frequency
uint32_t baudrate;          //baudrate
uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
DRV_I2S_BUFFER_HANDLE bufferHandle;
APP_DATA_S state;           //Application specific state
uintptr_t contextHandle;

void SYS_Initialize ( void* data )
{
    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
}


void App_Task(void)
{
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;

            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);

                /* Set the Event handler */
                DRV_I2S_BufferEventHandlerSet(handle,App_BufferEventHandler,
                contextHandle);

                /* Add a buffer to write*/
                DRV_I2S_BufferAddWrite(handle, &bufferHandle
                                        myAudioBuffer, BUFFER_SIZE);
                if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
                {
                    // Error handling here
```

```
                }
                state = APP_STATE_IDLE;
            }
            break;

            case APP_STATE_WAIT_FOR_DONE:
                state = APP_STATE_DONE;
            break;

            case  APP_STATE_DONE:
                // Close done
                DRV_I2S_Close(handle);
            break;

            case  APP_STATE_IDLE:
                // Do nothing
            break;

            default:
            break;
        }
}


void App_BufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    uint8_t temp;

    if(DRV_I2S_BUFFER_EVENT_COMPLETE == event)
    {
        // Can set state = APP_STATE_WAIT_FOR_DONE;
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ERROR == event)
    {
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ABORT == event)
    {
        // Take Action as needed
    }
    else
    {
        // Do nothing
    }
}


void SYS_Tasks ( void )
{
    DRV_I2S_Tasks((SYS_MODULE_OBJ)sysObj);
    DRV_I2S_TasksError((SYS_MODULE_OBJ)sysObj);

    /* Call the application's tasks routine */
    APP_Tasks ( );
}
```

Example 2:

```
// The following is an example for interrupt mode buffered transmit

#define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
#define BUFFER_SIZE 1000
// I2S initialization structure.
// This should be populated with necessary settings.
// attributes dmaChannelTransmit/dmaChannelReceive
// and dmaInterruptTransmitSource/dmaInterruptReceiveSource
// must be set if DMA mode of operation is desired.
```

```
DRV_I2S_INIT i2sInit;
SYS_MODULE_OBJ sysObj; //I2S module object
DRV_HANDLE handle;  //Client handle
uint32_t i2sClock;   //BCLK frequency
uint32_t baudrate;   //baudrate
uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
DRV_I2S_BUFFER_HANDLE bufferHandle;
APP_DATA_S state; //Application specific state
uintptr_t contextHandle;

void SYS_Initialize ( void* data )
{
    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
}


void App_Task(void)
{
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;

            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);

                /* Set the Event handler */
                DRV_I2S_BufferEventHandlerSet(handle,App_BufferEventHandler,
                contextHandle);

                /* Add a buffer to write*/
                DRV_I2S_BufferAddWrite(handle, &bufferHandle
                                       myAudioBuffer, BUFFER_SIZE);
                if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
                {
                    // Error handling here
                }
                state = APP_STATE_IDLE;
            }
            break;

            case APP_STATE_WAIT_FOR_DONE:

                state = APP_STATE_DONE;
            break;


            case  APP_STATE_DONE:
            {
```

```
                // Close done
                DRV_I2S_Close(handle);
            }
            break;

            case  APP_STATE_IDLE:
                // Do nothing
            break;

            default:
            break;
        }
}


void App_BufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    uint8_t temp;

    if(DRV_I2S_BUFFER_EVENT_COMPLETE == event)
    {
        // Can set state = APP_STATE_WAIT_FOR_DONE;
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ERROR == event)
    {
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ABORT == event)
    {
        // Take Action as needed
    }
    else
    {
        // Do nothing
    }
}


void SYS_Tasks ( void )
{
    /* Call the application's tasks routine */
    APP_Tasks ( );
}


void __ISR ( _SPI1_VECTOR ) _InterruptHandler_I2S1 ( void )
{
    // Call the "tasks" functions for I2S module
    DRV_I2S_Tasks((SYS_MODULE_OBJ)sysObj);
    DRV_I2S_TasksError((SYS_MODULE_OBJ)sysObj);
}


// If DMA Channel 1 was setup during initialization instead of the previous I2S ISR, the following should
be implemented
void __ISR ( _DMA1_VECTOR ) _InterruptHandler_DMA_CHANNEL_1 ( void )
{
    // Call the DMA system tasks which internally will call the I2S Tasks.
    SYS_DMA_Tasks((SYS_MODULE_OBJ)sysObj);
    SYS_DMA_TasksError((SYS_MODULE_OBJ)sysObj);
}
```

Client Operations - Non-buffered

This section provides information on non-buffered client operations.

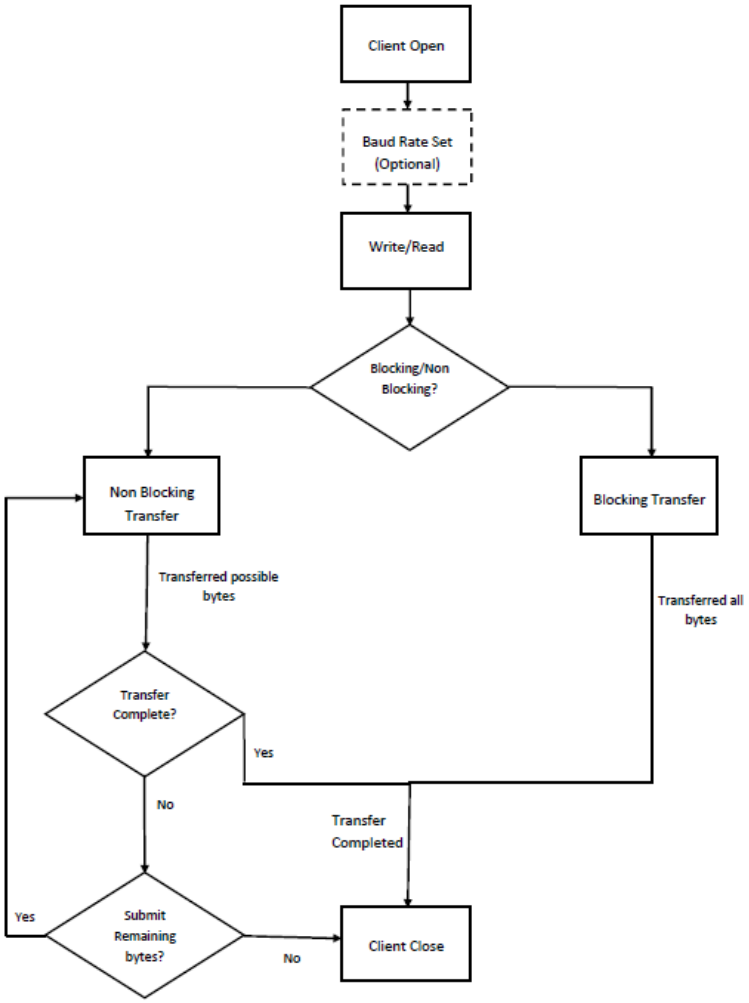
Description

Client Operations - Non-buffered

Client non-buffered operations provide a basic interface for the driver operation. This interface could be used by applications which have do not have buffered data transfer requirements. The functions DRV_I2S_Read and DRV_I2S_Write are the non-buffered data operation functions. The non-buffered functions are blocking/non-blocking depending upon the mode (ioIntent) the client was opened. If the client was opened for blocking mode these functions will only return when (or will block until) the specified data operation is completed or if an error occurred. If the client was opened for non-blocking mode, these functions will return with the number of bytes that were actually accepted for operation. The function will not wait until the data operation has completed.

Note: Non-buffered functions do not support interrupt/DMA mode.

The following diagram illustrates the non-buffered data operations



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the non-buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. The necessary ports setup and remapping must be done for I2S lines: ADCDAT, DACDAT, BCLK, LRCK and MCLK (if required).
3. The driver object should have been initialized by calling DRV_I2S_Initialize.
4. Open the driver using DRV_I2S_Open with the necessary ioIntent to get a client handle.
5. The necessary BCLK, LRCK, and MCLK should be set up so as to generate the required media bit rate.
6. The necessary Baud rate value should be set up by calling DRV_I2S_BaudrateSet.
7. The Transmit/Receive data should be set up by calling DRV_I2S_Write/DRV_I2S_Read.
8. Repeat step 5 through step 7 to handle multiple buffer transmission and reception.

9. When the client is done it can use DRV_I2S_Close to close the client handle.

Example 1:

```
// The following is an example for a blocking transmit
    #define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
    #define BUFFER_SIZE 1000
    DRV_I2S_INIT i2sInit; //I2S initialization structure
                          //This should be populated with necessary settings
    SYS_MODULE_OBJ sysObj; //I2S module object
    APP_DATA_S state; //Application specific state
    DRV_HANDLE handle;  //Client handle
    uint32_t i2sClock;   //BCLK frequency
    uint32_t baudrate;   //baudrate
    uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
    uint32_t count;

    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for
    // I2S lines ADCDAT, DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
    while(1)
    {
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE | DRV_IO_INTENT_BLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;
            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);
                // Blocks here and transfer the buffer
                count = DRV_I2S_Write(handle, &myAudioBuffer,BUFFER_SIZE);
                if(count == DRV_I2S_WRITE_ERROR)
                {
                    //Handle Error
                } else
                {
                    // Transfer Done
                    state = APP_STATE_DONE;
                }
            }
            break;
            case  APP_STATE_DONE:
            {
                // Close done
                DRV_I2S_Close(handle);
            }
            break;
            default:
            break;
        }
    }
```

Example 2:

```
// Following is an example for a non blocking transmit
    #define SYS_I2S_DRIVER_INDEX DRV_I2S_1 //I2S Uses SPI Hardware
    #define BUFFER_SIZE 1000
    DRV_I2S_INIT i2sInit; //I2S initialization structure.
                          // This should be populated with necessary settings
    SYS_MODULE_OBJ sysObj; //I2S module object
    APP_DATA_S state; //Application specific state
    DRV_HANDLE handle;  //Client handle
    uint32_t i2sClock;   //BCLK frequency
    uint32_t baudrate;   //baudrate
    uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
    uint32_t count,total,size;

    total = 0;
    size = BUFFER_SIZE;

    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }

    while(1)
    {
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;
            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);
                // Transfer whatever possible number of bytes
                count = DRV_I2S_Write(handle, &myAudioBuffer,size);
                if(count == DRV_I2S_WRITE_ERROR)
                {
                    //Handle Error
                } else
                {
                    // 'count' bytes transferred
                    state = APP_STATE_WAIT_FOR_DONE;
                }
            }
            break;
            case APP_STATE_WAIT_FOR_DONE:
                // Can perform other Application tasks here
                // ........................
                // ........................
                // ........................
                size = size - count;
                if(size!=0)
                {
```

```
                    // Change the state so as to submit
                    // another possible transmission
                    state = APP_STATE_WAIT_FOR_READY;
                }
                else
                {
                    // We are done
                    state = APP_STATE_DONE;
                }
            break;
            case  APP_STATE_DONE:
            {
                if(DRV_I2S_CLOSE_FAILURE == DRV_I2S_Close(handle))
                {
                    // Handle error
                }
                else
                {
                    // Close done
                }
            }
            break;
            default:
            break;
        }
    }
```

Configuring the Library

Client Configuration

	Name	Description
	DRV_I2S_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_I2S_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

System Configuration

	Name	Description
	DRV_I2S_INDEX	I2S Static Index selection
	DRV_I2S_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2S_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2S_INTERRUPT_SOURCE_ERROR	Defines the interrupt source for the error interrupt
	DRV_I2S_INTERRUPT_SOURCE_RECEIVE	Macro to define the Receive interrupt source in case of static driver
	DRV_I2S_INTERRUPT_SOURCE_TRANSMIT	Macro to define the Transmit interrupt source in case of static driver
	DRV_I2S_PERIPHERAL_ID	Configures the I2S PLIB Module ID
	DRV_I2S_RECEIVE_DMA_CHANNEL	Macro to defines the I2S Driver Receive DMA Channel in case of static driver
	DRV_I2S_STOP_IN_IDLE	Identifies whether the driver should stop operations in stop in Idle mode.
	DRV_I2S_TRANSMIT_DMA_CHANNEL	Macro to defines the I2S Driver Transmit DMA Channel in case of static driver
	DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL	Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver

Description

The configuration of the I2S Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the I2S Driver Library. Based on the selections made, the I2S Driver Library may support the selected features. These configuration settings will apply to all instances of the I2S Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

System Configuration

DRV_I2S_INDEX Macro

I2S Static Index selection

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INDEX
```

Description

Index - Used for static drivers

I2S Static Index selection for the driver object reference. This macro defines the driver index in case of static and static multi-client build. For example, if this macro is set to DRV_I2S_INDEX_2, then static driver APIs would be DRV_I2S2_Initialize(), DRV_I2S2_Open() etc. When building static drivers, this macro should be different for each static build of the I2S driver that needs to be included in the project.

Remarks

This index is required to make a reference to the driver object

DRV_I2S_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INSTANCES_NUMBER
```

Description

I2S driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of I2S modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None

DRV_I2S_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_MODE
```

Description

I2S Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values it can take are

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, then Interrupt Service Routine for the interrupt should be defined in the application. The DRV_I2S_Tasks() routine should be called in the ISR.

DRV_I2S_INTERRUPT_SOURCE_ERROR Macro

Defines the interrupt source for the error interrupt

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_ERROR
```

Description

Error Interrupt Source

Macro to define the Error interrupt source in case of static driver. The interrupt source defined by this macro will override the errorInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module error interrupt enumeration in the Interrupt PLIB for the microcontroller.

DRV_I2S_INTERRUPT_SOURCE_RECEIVE Macro

Macro to define the Receive interrupt source in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_RECEIVE
```

Description

Receive Interrupt Source

Macro to define the Receive interrupt source in case of static driver. The interrupt source defined by this macro will override the rxInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module receive interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_I2S_INTERRUPT_SOURCE_TRANSMIT Macro

Macro to define the Transmit interrupt source in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_TRANSMIT
```

Description

Transmit Interrupt Source

Macro to define the TX interrupt source in case of static driver. The interrupt source defined by this macro will override the txInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module transmit interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_I2S_PERIPHERAL_ID Macro

Configures the I2S PLIB Module ID

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_PERIPHERAL_ID
```

Description

I2S Peripheral Library Module ID

This macro configures the PLIB ID if the driver is built statically. This value will override the I2SID member of the DRV_I2S_INIT initialization data structure. In that when the driver is built statically, the I2SID member of the DRV_I2S_INIT data structure will be ignored by the driver initialization routine and this macro will be considered. This should be set to the PLIB ID of I2S module (I2S_ID_1, I2S_ID_2 and so on).

DRV_I2S_RECEIVE_DMA_CHANNEL Macro

Macro to defines the I2S Driver Receive DMA Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_RECEIVE_DMA_CHANNEL
```

Description

I2S Driver Receive DMA Channel

Macro to define the I2S Receive DMA Channel in case of static driver. The DMA channel defined by this macro will override the dmaChannelReceive member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_I2S_STOP_IN_IDLE Macro

Identifies whether the driver should stop operations in stop in Idle mode.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_STOP_IN_IDLE
```

Description

I2S driver objects configuration

Identifies whether the driver should stop operations in stop in Idle mode. true - Indicates stop in idle mode. false - Indicates do not stop in Idle mode.

Remarks

None

DRV_I2S_TRANSMIT_DMA_CHANNEL Macro

Macro to defines the I2S Driver Transmit DMA Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_TRANSMIT_DMA_CHANNEL
```

Description

I2S Driver Transmit DMA Channel

Macro to define the I2S Transmit DMA Channel in case of static driver. The DMA channel defined by this macro will override the dmaChannelTransmit member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL Macro

Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL
```

Description

I2S Driver Receive DMA Chaining Channel

Macro to define the I2S Receive DMA Chaining Channel in case of static driver. The DMA channel defined by this macro will override the dmaChaningChannelReceive member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

Client Configuration

DRV_I2S_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_CLIENTS_NUMBER
```

Description

I2S Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if I2S1 will be accessed by 2 clients and I2S2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_I2S_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_I2S_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None

DRV_I2S_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_QUEUE_DEPTH_COMBINED
```

Description

I2S Driver Buffer Queue Entries

This macro defined the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_I2S_BufferAddWrite() function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_I2S_BufferAddRead() function.

A buffer queue will contains buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer

entries that will be available for use between all I2S driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

As an example, consider the case of a dynamic driver (say 2 instances) where instance 1 will queue up to 3 write requests and up to 2 read requests, and instance 2 will queue up to 2 write requests and up to 6 read requests, the value of this macro should be 13 (2 + 3 + 2 + 6).

Remarks

The maximum combined queue depth should not be greater than 0xFFFF(ie 65535)

Building the Library

This section lists the files that are available in the I2S Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/i2s`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_i2s.h	This file provides the interface definitions of the I2S driver.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_i2s_dma.c	This file contains the core implementation of the I2S driver with DMA support.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/dynamic/drv_i2s_dma_advanced.c	This file contains the implementation of the I2S driver with DMA support using the channel chaining feature.
/src/dynamic/drv_i2s.c	This file contains the implementation of the I2S driver without DMA support.
/src/dynamic/drv_i2s_read_write.c	This file contains the basic read/write implementation of the I2S driver.

Module Dependencies

The I2S Driver Library depends on the following modules:

- SPI Peripheral Library
- DMA Peripheral Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_I2S_Deinitialize	Deinitializes the specified instance of the I2S driver module. **Implementation:** Dynamic

	DRV_I2S_Initialize	Initializes hardware and data for the instance of the I2S module. **Implementation:** Dynamic
	DRV_I2S_Status	Gets the current status of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_Tasks	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_I2S_Close	Closes an opened-instance of the I2S driver. **Implementation:** Dynamic
	DRV_I2S_Open	Opens the specified I2S driver instance and returns a handle to it. **Implementation:** Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_I2S_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_I2S_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2S_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_I2S_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_I2S_Read	Reads data from the I2S. **Implementation:** Dynamic
	DRV_I2S_Write	Writes data to the I2S. **Implementation:** Dynamic
	DRV_I2S_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic

d) Miscellaneous Functions

	Name	Description
	DRV_I2S_BaudSet	This function sets the baud. **Implementation:** Dynamic
	DRV_I2S_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Dynamic
	DRV_I2S_ReceiveErrorIgnore	This function enable/disable ignoring of the receive overflow error. **Implementation:** Dynamic
	DRV_I2S_TransmitErrorIgnore	This function enable/disable ignoring of the transmit underrun error. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_I2S_AUDIO_PROTOCOL_MODE	Identifies the Audio Protocol Mode of the I2S module.
	DRV_I2S_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_I2S_BUFFER_EVENT_HANDLER	Pointer to a I2S Driver Buffer Event handler function
	DRV_I2S_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_I2S_CLOCK_MODE	Identifies the various clock modes of the I2S module.
	DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data

	DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	DRV_I2S_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_I2S_MODE	Identifies the usage modes of the I2S module.
	DRV_I2S_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_I2S_COUNT	Number of valid I2S driver indices
	DRV_I2S_READ_ERROR	I2S Driver Read Error.
	DRV_I2S_WRITE_ERROR	I2S Driver Write Error.
	DRV_I2S_INDEX_0	I2S driver index definitions
	DRV_I2S_INDEX_1	This is macro DRV_I2S_INDEX_1.
	DRV_I2S_INDEX_2	This is macro DRV_I2S_INDEX_2.
	DRV_I2S_INDEX_3	This is macro DRV_I2S_INDEX_3.
	DRV_I2S_INDEX_4	This is macro DRV_I2S_INDEX_4.
	DRV_I2S_INDEX_5	This is macro DRV_I2S_INDEX_5.
	DRV_I2S_INTERFACE	This structure defines a structure of I2S Driver function pointers.

Description

This section describes the Application Programming Interface (API) functions of the I2S Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_I2S_Deinitialize Function

Deinitializes the specified instance of the I2S driver module.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the I2S driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_I2S_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_I2S_Initialize
SYS_STATUS          status;


DRV_I2S_Deinitialize(object);

status = DRV_I2S_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_I2S_Initialize routine

Function

void DRV_I2S_Deinitialize(SYS_MODULE_OBJ object)

DRV_I2S_Initialize Function

Initializes hardware and data for the instance of the I2S module.

Implementation: Dynamic

File

drv_i2s.h

C

```
SYS_MODULE_OBJ DRV_I2S_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT *const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the I2S driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the I2S module ID. For example, driver instance 0 can be assigned to I2S2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_I2S_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other I2S routine is called.

This routine should only be called once during system initialization unless DRV_I2S_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

To Enable the DMA mode of operation the init parameters 'dmaChannelTransmit' /'dmaChannelReceive' must be set to valid DMA channel. When DMA mode of operation is enabled, the normal mode(Usual TX and RX) operation is inhibited. When 'dmaChannelTransmit'/'dmaChannelReceive' are set to valid channel numbers the related DMA interrupt source parameters 'dmaInterruptTransmitSource'/ 'dmaInterruptReceiveSource' must be set with appropriate DMA channel interrupt source.

Preconditions

If DMA mode of operation is intended, SYS_DMA_Initialize should have been called before calling this function.

Example

```
DRV_I2S_INIT               init;
SYS_MODULE_OBJ             objectHandle;

init.moduleInit.value          = SYS_MODULE_POWER_RUN_FULL;
init.spiID                     = SPI_ID_1;
init.usageMode                 = DRV_I2S_MODE_MASTER;
init.baudClock                 = SPI_BAUD_RATE_MCLK_CLOCK;
init.baud                      = 48000;
init.clockMode                 = DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL;
init.audioCommWidth            = SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL;
init.audioTransmitMode         = SPI_AUDIO_TRANSMIT_STEREO;
init.inputSamplePhase          = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
init.protocolMode              = DRV_I2S_AUDIO_I2S;
init.txInterruptSource         = INT_SOURCE_SPI_1_TRANSMIT;
init.rxInterruptSource         = INT_SOURCE_SPI_1_RECEIVE;
init.errorInterruptSource      = INT_SOURCE_SPI_1_ERROR;
init.queueSizeTransmit         = 3;
init.queueSizeReceive          = 2;
init.dmaChannelTransmit        = DMA_CHANNEL_NONE;
init.dmaChannelReceive         = DMA_CHANNEL_NONE;

objectHandle = DRV_I2S_Initialize(DRV_I2S_INDEX_1, (SYS_MODULE_INIT*)init);
```

```
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_I2S_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT *const init

);

DRV_I2S_Status Function

Gets the current status of the I2S driver module.

Implementation: Dynamic

File

drv_i2s.h

C

```
SYS_STATUS DRV_I2S_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the I2S driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_I2S_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize
SYS_STATUS          i2sStatus;

i2sStatus = DRV_I2S_Status(object);
if (SYS_STATUS_READY == i2sStatus)
{
    // This means the driver can be opened using the
    // DRV_I2S_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_I2S_Initialize routine

Function

SYS_STATUS DRV_I2S_Status(SYS_MODULE_OBJ object)

DRV_I2S_Tasks Function

Maintains the driver's receive state machine and implements its ISR.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal receive state machine and implement its transmit and receive ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called from the interrupt service routine of the I2S that is associated with this I2S driver hardware instance.

In DMA mode of operation, this function should be called from the interrupt service routine of the channel associated with the transmission/reception of the I2s driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize

while (true)
{
    DRV_I2S_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_I2S_Initialize)

Function

void DRV_I2S_Tasks(SYS_MODULE_OBJ object)

DRV_I2S_TasksError Function

Maintains the driver's error state machine and implements its ISR.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_TasksError(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal error state machine and implement its error ISR for interrupt-driven implementations. In polling

mode, this function should be called from the SYS_Tasks() function. In interrupt mode, this function should be called in the error interrupt service routine of the I2S that is associated with this I2S driver hardware instance.

In DMA mode of operation, this function should be called from the interrupt service routine of the channel associated with the transmission/reception of the I2s driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize

while (true)
{
    DRV_I2S_TasksError (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_I2S_Initialize)

Function

void DRV_I2S_TasksError (SYS_MODULE_OBJ object)

b) Client Setup Functions

DRV_I2S_Close Function

Closes an opened-instance of the I2S driver.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Close(const DRV_HANDLE handle);
```

Returns

- None

Description

This routine closes an opened-instance of the I2S driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_I2S_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_I2S_Open
```

```
DRV_I2S_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2S_Close(DRV_Handle handle)

DRV_I2S_Open Function

Opens the specified I2S driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_i2s.h

C

```
DRV_HANDLE DRV_I2S_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_I2S_CLIENTS_NUMBER is

insufficient.

- if the client is trying to open the driver but driver has been opened

exclusively by another client.

- if the driver hardware instance being opened is not initialized or is

invalid.

Description

This routine opens the specified I2S driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_I2S_Read() and DRV_I2S_Write() functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_I2S_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_I2S_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_I2S_Open(DRV_I2S_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

```
  DRV_HANDLE DRV_I2S_Open
(
const SYS_MODULE_INDEX drvIndex,
const    DRV_IO_INTENT    ioIntent
)
```

c) Data Transfer Functions

DRV_I2S_BufferAddRead Function

Schedule a non-blocking driver read operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddRead(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void * buffer,
size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for write-only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;
```

```
// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as returned by the DRV_I2S_Open function
buffer	Buffer where the received data will be stored.
size	Buffer size in bytes
bufferHandle	Pointer to an argument that will contain the return buffer handle

Function

void DRV_I2S_BufferAddRead

(

const DRV_HANDLE handle,

DRV_I2S_BUFFER_HANDLE *bufferHandle,

void * buffer, size_t size

)

DRV_I2S_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddWrite(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void * buffer,
size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddWrite(myI2Shandle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:
```

```
            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as return by the DRV_I2S_Open function
buffer	Data to be transmitted
size	Buffer size in bytes
bufferHandle	Pointer to an argument that will contain the return buffer handle

Function

void DRV_I2S_BufferAddWrite

(

const DRV_HANDLE handle,

DRV_I2S_BUFFER_HANDLE *bufferHandle,

void * buffer, size_t size

);

DRV_I2S_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddWriteRead(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every I2S write. The transmit and receive size must be same.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddWriteRead(myI2Shandle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as returned by the DRV_I2S_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	Buffer where the transmit data will be stored
receiveBuffer	Buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_I2S_BufferAddWriteRead

(

```
const       DRV_HANDLE handle,
    DRV_I2S_BUFFER_HANDLE   *bufferHandle,
void *transmitBuffer, void *receiveBuffer,
size_t size
)
```

DRV_I2S_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_I2S_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandler,
                                (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE handle, uintptr_t contextHandle)
```

```
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_I2S_BufferEventHandlerSet

(

const DRV_HANDLE handle,

DRV_I2S_BUFFER_EVENT_HANDLER eventHandler,

uintptr_t contextHandle

)

DRV_I2S_BufferCombinedQueueSizeGet Function

This function returns the number of bytes queued (to be processed) in the buffer queue.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_BufferCombinedQueueSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired client handle.

Description

This function returns the number of bytes queued (to be processed) in the buffer queue of the driver instance associated with the calling client. The client can use this function to know number of remaining bytes (from the buffers submitted by it)is in the queue to be transmitted.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can get to know dynamically available data in the queue to be
// transmitted by calling DRV_I2S_BufferCombinedQueueSizeGet
bufferQueuedSize = DRV_I2S_BufferCombinedQueueSizeGet(myI2SHandle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

size_t DRV_I2S_BufferCombinedQueueSizeGet(DRV_HANDLE handle)

DRV_I2S_BufferQueueFlush Function

This function flushes off the buffers associated with the client object.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferQueueFlush(DRV_HANDLE handle);
```

Returns

None.

Description

This function flushes off the buffers associated with the client object and disables the DMA channel used for transmission.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can stop the data processing and flushoff the data
// in the queue by calling DRV_I2S_BufferQueueFlush
DRV_I2S_BufferQueueFlush(myI2SHandle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

void DRV_I2S_BufferQueueFlush(DRV_HANDLE handle)

DRV_I2S_Read Function

Reads data from the I2S.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_Read(const DRV_HANDLE handle, uint8_t * buffer, const size_t numBytes);
```

Returns

Number of bytes actually copied into the caller's buffer. Returns DRV_I2S_READ_ERROR in case of an error.

Description

This routine reads data from the I2S. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numBytes of bytes have been received or if an error occurred.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually read. The function will not wait until numBytes of bytes have been read.

Remarks

This function is thread safe in a RTOS application. It is recommended that this function not be called in I2S Driver Event Handler due to the potential blocking nature of the function. This function should not be called directly in an ISR. It should not be called in the event handler associated with another I2S driver instance.

This function does not supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
DRV_HANDLE      myI2SHandle;    // Returned from DRV_I2S_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_I2S_Read(myI2SHandle, &myBuffer[total],
                          MY_BUFFER_SIZE - total);
    total += count;
    if(count == DRV_I2S_READ_ERROR)
    {
        // Handle error ...
    }
    else
    {
        // Do what needs to be..
    }
} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer into which the data read from the I2S instance will be placed.
numBytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Function

size_t DRV_I2S_Read

(

const DRV_HANDLE handle,

uint8_t *buffer,

const size_t numBytes

)

DRV_I2S_Write Function

Writes data to the I2S.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_Write(const DRV_HANDLE handle, uint8_t * buffer, const size_t numBytes);
```

Returns

Number of bytes actually written to the driver. Return DRV_I2S_WRITE_ERROR in case of an error.

Description

This routine writes data to the I2S. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been transmitted or if an error occurred.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually accepted for transmission. The function will not wait until numBytes of bytes have been transmitted.

Remarks

This function is thread safe in a RTOS application. It is recommended that this function not be called in I2S Driver Event Handler due to the potential blocking nature of the function. This function should not be called directly in an ISR. It should not be called in the event handler associated with another USART driver instance.

This function does not supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
DRV_HANDLE myI2SHandle;    // Returned from DRV_I2S_Open
char myBuffer[MY_BUFFER_SIZE];
int count;
unsigned int total;
total = 0;
do
{
    count  = DRV_I2S_Write(myI2SHandle, &myBuffer[total],
            MY_BUFFER_SIZE - total);
    total += count;
    if(count == DRV_I2S_WRITE_ERROR)
    {
        // Handle error ...
    }
    else
    {
        // Do what needs to be ..
    }
} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer containing the data to written.
numbytes	size of the buffer

Function

size_t DRV_I2S_Write

(

const DRV_HANDLE handle,

void * buffer,

const size_t numbytes

)

DRV_I2S_BufferProcessedSizeGet Function

This function returns number of bytes that have been processed for the specified buffer.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_BufferProcessedSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. If this function is called on a invalid buffer handle, or if the buffer handle has expired, the function returns 0.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(myI2SHandle);

            break;

        default:
```

```
            break;
        }
    }
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_I2S_BufferProcessedSizeGet(DRV_HANDLE handle)

d) Miscellaneous Functions

DRV_I2S_BaudSet Function

This function sets the baud.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BaudSet(DRV_HANDLE handle, uint32_t spiClock, uint32_t baud);
```

Returns

None

Description

This function sets the baud rate for the I2S operation.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t clock;


// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

// Sets the baud rate to a new value as below
baud = 115200;
clock = 40000000UL;
DRV_I2S_BaudSet(myI2SHandle, clock, baud);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);
```

```
if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(myI2SHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
i2sClock	The Source clock frequency to the i2S module.
baud	The baud to be set.

Function

void DRV_I2S_BaudSet(DRV_HANDLE handle, uint32_t spiClock, uint32_t baud)

DRV_I2S_ErrorGet Function

This function returns the error(if any) associated with the last client request.

Implementation: Dynamic

File

drv_i2s.h

C

```
DRV_I2S_ERROR DRV_I2S_ErrorGet(DRV_HANDLE handle);
```

Returns

A DRV_I2S_ERROR type indicating last known error status.

Description

This function returns the error(if any) associated with the last client request. The DRV_I2S_Read() and DRV_I2S_Write() will update the client error status when these functions return DRV_I2S_READ_ERROR and DRV_I2S_WRITE_ERROR, respectively. If the driver send a

DRV_I2S_BUFFER_EVENT_ERROR to the client, the client can call this function to know the error cause. The error status will be updated on every operation and should be read frequently (ideally immediately after the driver operation has completed) to know the relevant error status.

Remarks

It is the client's responsibility to make sure that the error status is obtained frequently. The driver will update the client error status regardless of whether this has been examined by the client.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet( myI2SHandle, APP_I2SBufferEventHandle,
                                (uintptr_t)&myAppObj );

DRV_I2S_BufferAddRead( myI2Shandle,&bufferHandle,
                                     myBuffer, MY_BUFFER_SIZE );

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler( DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_SUCCESS:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_FAILURE:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred. We can also find
            // the error cause.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(myI2SHandle);
            if(DRV_I2S_ERROR_RECEIVE_OVERRUN == DRV_I2S_ErrorGet(myI2SHandle))
            {
                // There was an receive over flow error.
                // Do error handling here.
            }

            break;
```

```
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

DRV_I2S_ERROR DRV_I2S_ErrorGet(DRV_HANDLE handle)

DRV_I2S_ReceiveErrorIgnore Function

This function enable/disable ignoring of the receive overflow error.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_ReceiveErrorIgnore(DRV_HANDLE handle, bool errorEnable);
```

Returns

None

Description

A receive overflow is not a critical error; during receive overflow data in the FIFO is not overwritten by receive data. Ignore receive overflow is needed for cases when there is a general performance problem in the system that software must handle properly.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t baud;*

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

// Enable ignoring of receive overflow error
DRV_I2S_ReceiveErrorIgnore(myI2SHandle, true);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}
```

```
// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
errorIgnore	When set to 'true' enables ignoring of transmit underrun error. When set to 'false' disables ignoring of transmit underrun error.

Function

void DRV_I2S_ReceiveErrorIgnore(DRV_HANDLE handle, bool errorEnable)

DRV_I2S_TransmitErrorIgnore Function

This function enable/disable ignoring of the transmit underrun error.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_TransmitErrorIgnore(DRV_HANDLE handle, bool errorIgnore);
```

Returns

None

Description

A Transmit underrun error is not a critical error and zeros are transmitted until the SPIxTXB is not empty. Ignore Transmit underrun error is needed for cases when software does not care or does not need to know about underrun conditions.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t baud;*

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

// Enable ignoring of transmit underrun error
DRV_I2S_TransmitErrorIgnore(myI2SHandle, true);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
errorIgnore	When set to 'true' enables ignoring of transmit underrun error. When set to 'false' disables ignoring of transmit underrun error.

Function

void DRV_I2S_TransmitErrorIgnore(DRV_HANDLE handle, bool errorIgnore)

e) Data Types and Constants

DRV_I2S_AUDIO_PROTOCOL_MODE Enumeration

Identifies the Audio Protocol Mode of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_AUDIO_I2S,
  DRV_I2S_AUDIO_LFET_JUSTIFIED,
  DRV_I2S_AUDIO_RIGHT_JUSTIFIED,
  DRV_I2S_AUDIO_PCM_DSP
} DRV_I2S_AUDIO_PROTOCOL_MODE;
```

Members

Members	Description
DRV_I2S_AUDIO_I2S	I2S Audio Protocol
DRV_I2S_AUDIO_LFET_JUSTIFIED	Left Justified Audio Protocol
DRV_I2S_AUDIO_RIGHT_JUSTIFIED	Right Justified Audio Protocol
DRV_I2S_AUDIO_PCM_DSP	PCM/DSP Audio Protocol

Description

I2S Audio Protocol Mode

This enumeration identifies Audio Protocol Mode of the I2S module.

Remarks

None.

DRV_I2S_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_BUFFER_EVENT_COMPLETE,
  DRV_I2S_BUFFER_EVENT_ERROR,
  DRV_I2S_BUFFER_EVENT_ABORT
} DRV_I2S_BUFFER_EVENT;
```

Members

Members	Description
DRV_I2S_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_I2S_BUFFER_EVENT_ERROR	Error while processing the request
DRV_I2S_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

I2S Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead functions.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_I2S_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_I2S_BUFFER_EVENT_HANDLER Type

Pointer to a I2S Driver Buffer Event handler function

File

drv_i2s.h

C

```
typedef void (* DRV_I2S_BUFFER_EVENT_HANDLER)(DRV_I2S_BUFFER_EVENT event, DRV_I2S_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

I2S Driver Buffer Event Handler Function

This data type defines the required function signature for the I2S driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_I2S_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_I2S_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_I2S_ErrorGet function can be called to know the error. The DRV_I2S_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_I2S_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite and DRV_I2S_BufferAddWriteRead functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running. For example, buffers cannot be added I2S2 driver in I2S1 driver event handler.

Example

```
void APP_MyBufferEventHandler
(
    DRV_I2S_BUFFER_EVENT event,
    DRV_I2S_BUFFER_HANDLE bufferHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // Handle the completed buffer.
```

```
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_I2S_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_i2s.h

C

```
typedef uintptr_t DRV_I2S_BUFFER_HANDLE;
```

Description

I2S Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite, and DRV_I2S_BufferAddWriteRead functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_I2S_CLOCK_MODE Enumeration

Identifies the various clock modes of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_RISE,
  DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
  DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL,
  DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_RISE
} DRV_I2S_CLOCK_MODE;
```

Members

Members	Description
DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_RISE	I2S Clock Mode 0 - Idle State Low & Sampling on Rising Edge
DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_FALL	I2S Clock Mode 1 - Idle State Low & Sampling on Falling Edge
DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL	I2S Clock Mode 2 - Idle State High & Sampling on Falling Edge
DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_RISE	I2S Clock Mode 3 - Idle State High & Sampling on Rising Edge

Description

I2S Clock Mode Selection

This enumeration identifies the supported clock modes of the I2S module.

Remarks

None.

DRV_I2S_DATA16 Structure

Defines the left and right channel data for 16-bit audio data

File

drv_i2s.h

C

```
typedef struct {
  int16_t leftData;
  int16_t rightData;
} DRV_I2S_DATA16;
```

Members

Members	Description
int16_t leftData;	Left channel data
int16_t rightData;	Right channel data

Description

I2S Driver Audio Data 16

Defines the left and right channel data for 16-bit audio data

Remarks

None.

DRV_I2S_DATA24 Structure

Defines the left and right channel data for 24-bit audio data

File

drv_i2s.h

C

```
typedef struct {
  int32_t leftData : 24;
  int32_t leftDataPad : 8;
  int32_t rightData : 24;
  int32_t rightDataPad : 8;
} DRV_I2S_DATA24;
```

Members

Members	Description
int32_t leftData : 24;	Left channel data
int32_t leftDataPad : 8;	Left channel data pad
int32_t rightData : 24;	Right channel data
int32_t rightDataPad : 8;	Right channel data pad

Description

I2S Driver Audio Data 24

Defines the left and right channel data for 24-bit audio data

Remarks

None.

DRV_I2S_DATA32 Structure

Defines the left and right channel data for 32-bit audio data

File

drv_i2s.h

C

```
typedef struct {
  int32_t leftData;
  int32_t rightDataPad;
} DRV_I2S_DATA32;
```

Members

Members	Description
int32_t leftData;	Left channel data
int32_t rightDataPad;	Right channel data

Description

I2S Driver Audio Data 32

Defines the left and right channel data for 32-bit audio data

Remarks

None.

DRV_I2S_ERROR Enumeration

Defines the possible errors that can occur during driver operation.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_ERROR_NONE,
  DRV_I2S_ERROR_RECEIVE_OVERFLOW,
  DRV_I2S_ERROR_TRANSMIT_UNDERUN,
  DRV_I2S_ERROR_FRAMING,
  DRV_I2S_ERROR_ADDRESS
} DRV_I2S_ERROR;
```

Members

Members	Description
DRV_I2S_ERROR_NONE	Data was transferred successfully.
DRV_I2S_ERROR_RECEIVE_OVERFLOW	Receive overflow error.
DRV_I2S_ERROR_TRANSMIT_UNDERUN	Transmit underrun error.
DRV_I2S_ERROR_FRAMING	Framing error.
DRV_I2S_ERROR_ADDRESS	Channel address error (Applicable in DMA mode)

Description

I2S Driver Error

This data type defines the possible errors that can occur when occur during USART driver operation. These values are returned by DRV_I2S_ErrorGet function.

Remarks

None.

DRV_I2S_MODE Enumeration

Identifies the usage modes of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_MODE_SLAVE,
  DRV_I2S_MODE_MASTER
} DRV_I2S_MODE;
```

Members

Members	Description
DRV_I2S_MODE_SLAVE	I2S Mode Slave
DRV_I2S_MODE_MASTER	I2S Mode Master

Description

I2S Usage Modes Enumeration

This enumeration identifies the whether the I2S module will be used as a master or slave.

Remarks

None.

DRV_I2S_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_i2s.h

C

```
#define DRV_I2S_BUFFER_HANDLE_INVALID ((DRV_I2S_BUFFER_HANDLE)(-1))
```

Description

I2S Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite and DRV_I2S_BufferAddWriteRead functions if the buffer add request was not successful.

Remarks

None

DRV_I2S_COUNT Macro

Number of valid I2S driver indices

File

drv_i2s.h

C

```
#define DRV_I2S_COUNT
```

Description

I2S Driver Module Count

This constant identifies the maximum number of I2S Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of I2S instances on this microcontroller.

Remarks

This value is part-specific.

DRV_I2S_READ_ERROR Macro

I2S Driver Read Error.

File

drv_i2s.h

C

```
#define DRV_I2S_READ_ERROR ((size_t)(-1))
```

Description

I2S Driver Read Error

This constant is returned by DRV_I2S_Read function when an error occurs.

Remarks

None.

DRV_I2S_WRITE_ERROR Macro

I2S Driver Write Error.

File

drv_i2s.h

C

```
#define DRV_I2S_WRITE_ERROR ((size_t)(-1))
```

Description

I2S Driver Write Error

This constant is returned by DRV_I2S_Write() function when an error occurs.

Remarks

None.

DRV_I2S_INDEX_0 Macro

I2S driver index definitions

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_0 0
```

Description

Driver I2S Module Index

These constants provide I2S driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_I2S_Initialize and DRV_I2S_Open routines to identify the driver instance in use.

DRV_I2S_INDEX_1 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_1 1
```

Description

This is macro DRV_I2S_INDEX_1.

DRV_I2S_INDEX_2 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_2 2
```

Description

This is macro DRV_I2S_INDEX_2.

DRV_I2S_INDEX_3 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_3 3
```

Description

This is macro DRV_I2S_INDEX_3.

DRV_I2S_INDEX_4 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_4 4
```

Description

This is macro DRV_I2S_INDEX_4.

DRV_I2S_INDEX_5 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_5 5
```

Description

This is macro DRV_I2S_INDEX_5.

DRV_I2S_INTERFACE Structure

This structure defines a structure of I2S Driver function pointers.

File

drv_i2s.h

C

```
typedef struct {
  SYS_MODULE_OBJ (* initialize)(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
  void (* deinitialize)(SYS_MODULE_OBJ);
  SYS_STATUS (* status)(SYS_MODULE_OBJ object);
  void (* tasks)(SYS_MODULE_OBJ object);
  void (* tasksError)(SYS_MODULE_OBJ object);
  DRV_HANDLE (* open)(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

```
  void (* close)(const DRV_HANDLE handle);
  void (* bufferAddWrite)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void *buffer,
size_t size);
  void (* bufferAddRead)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void *buffer, size_t
size);
  void (* bufferAddWriteRead)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void
*transmitBuffer, void *receiveBuffer, size_t size);
  size_t (* read)(const DRV_HANDLE handle, uint8_t *buffer, const size_t numBytes);
  size_t (* write)(const DRV_HANDLE handle, uint8_t *buffer, const size_t numBytes);
  void (* eventHandlerSet)(DRV_HANDLE handle, const DRV_I2S_BUFFER_EVENT_HANDLER eventHandler, const
uintptr_t contextHandle);
  size_t (* bufferProcessedSizeGet)(DRV_HANDLE handle);
  size_t (* bufferCombinedQueueSizeGet)(DRV_HANDLE handle);
  void (* bufferQueueFlush)(DRV_HANDLE handle);
  DRV_I2S_ERROR (* errorGet)(DRV_HANDLE handle);
  void (* baudSet)(DRV_HANDLE handle, uint32_t peripheralClock, uint32_t baud);
  void (* setAudioCommunicationMode)(DRV_HANDLE handle, uint8_t audioCommWidth);
  void (* transmitErrorIgnore)(DRV_HANDLE handle, bool errorIgnore);
  void (* receiveErrorIgnore)(DRV_HANDLE handle, bool errorEnable);
} DRV_I2S_INTERFACE;
```

Members

Members	Description
SYS_MODULE_OBJ (* initialize)(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);	Pointer to the driver Initialization function
void (* deinitialize)(SYS_MODULE_OBJ);	Pointer to the driver Deinitialization function
SYS_STATUS (* status)(SYS_MODULE_OBJ object);	Pointer to the driver Status function
void (* tasks)(SYS_MODULE_OBJ object);	Pointer to the Tasks function
void (* tasksError)(SYS_MODULE_OBJ object);	Pointer to the Error Tasks function
DRV_HANDLE (* open)(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);	Pointer to the Open function
void (* close)(const DRV_HANDLE handle);	Pointer to the Close function
void (* bufferAddWrite)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void *buffer, size_t size);	Pointer to the Buffer Add Write function
void (* bufferAddRead)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void *buffer, size_t size);	Pointer to the Buffer Add Read function
void (* bufferAddWriteRead)(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE *bufferHandle, void *transmitBuffer, void *receiveBuffer, size_t size);	Pointer to the buffer Add Read Write function
size_t (* read)(const DRV_HANDLE handle, uint8_t *buffer, const size_t numBytes);	Pointer to the driver Read function
size_t (* write)(const DRV_HANDLE handle, uint8_t *buffer, const size_t numBytes);	Pointer to the driver Write function
void (* eventHandlerSet)(DRV_HANDLE handle, const DRV_I2S_BUFFER_EVENT_HANDLER eventHandler, const uintptr_t contextHandle);	Pointer to the driver Buffer Event Handler Set function
size_t (* bufferProcessedSizeGet)(DRV_HANDLE handle);	Pointer to the driver Buffer Processed Size Get function
size_t (* bufferCombinedQueueSizeGet)(DRV_HANDLE handle);	Pointer to the driver Buffer Combined Queue Size Get Function
void (* bufferQueueFlush)(DRV_HANDLE handle);	Pointer to the driver Buffer Queue Flush Function
DRV_I2S_ERROR (* errorGet)(DRV_HANDLE handle);	Pointer to the driver Error Get function
void (* baudSet)(DRV_HANDLE handle, uint32_t peripheralClock, uint32_t baud);	Pointer to the driver Baud Set function
void (* setAudioCommunicationMode)(DRV_HANDLE handle, uint8_t audioCommWidth);	Pointer to the driver Set Audio Communication mode function

void (* transmitErrorIgnore)(DRV_HANDLE handle, bool errorIgnore);	Pointer to the driver Transmit Error Ignore function
void (* receiveErrorIgnore)(DRV_HANDLE handle, bool errorEnable);	Pointer to the driver Receive Error Ignore function

Description

I2S Driver Interface

This structure defines a structure of I2S Driver function pointers. A driver of any peripheral that supports the I2S protocol can export such a structure. The top level I2S Driver abstraction layer will then use this structure to map a I2S Driver call to underlying peripheral driver.

Remarks

None.

Files

Files

Name	Description
drv_i2s.h	I2S Driver Interface header file
drv_i2s_config_template.h	I2S Driver Configuration Template.

Description

drv_i2s.h

I2S Driver Interface header file

Enumerations

	Name	Description
	DRV_I2S_AUDIO_PROTOCOL_MODE	Identifies the Audio Protocol Mode of the I2S module.
	DRV_I2S_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_I2S_CLOCK_MODE	Identifies the various clock modes of the I2S module.
	DRV_I2S_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_I2S_MODE	Identifies the usage modes of the I2S module.

Functions

	Name	Description
	DRV_I2S_BaudSet	This function sets the baud. **Implementation:** Dynamic
	DRV_I2S_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_I2S_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_I2S_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2S_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic
	DRV_I2S_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_I2S_Close	Closes an opened-instance of the I2S driver. **Implementation:** Dynamic
	DRV_I2S_Deinitialize	Deinitializes the specified instance of the I2S driver module. **Implementation:** Dynamic

	DRV_I2S_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Dynamic
	DRV_I2S_Initialize	Initializes hardware and data for the instance of the I2S module. **Implementation:** Dynamic
	DRV_I2S_Open	Opens the specified I2S driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_I2S_Read	Reads data from the I2S. **Implementation:** Dynamic
	DRV_I2S_ReceiveErrorIgnore	This function enable/disable ignoring of the receive overflow error. **Implementation:** Dynamic
	DRV_I2S_Status	Gets the current status of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_Tasks	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TransmitErrorIgnore	This function enable/disable ignoring of the transmit underrun error. **Implementation:** Dynamic
	DRV_I2S_Write	Writes data to the I2S. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_I2S_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_I2S_COUNT	Number of valid I2S driver indices
	DRV_I2S_INDEX_0	I2S driver index definitions
	DRV_I2S_INDEX_1	This is macro DRV_I2S_INDEX_1.
	DRV_I2S_INDEX_2	This is macro DRV_I2S_INDEX_2.
	DRV_I2S_INDEX_3	This is macro DRV_I2S_INDEX_3.
	DRV_I2S_INDEX_4	This is macro DRV_I2S_INDEX_4.
	DRV_I2S_INDEX_5	This is macro DRV_I2S_INDEX_5.
	DRV_I2S_READ_ERROR	I2S Driver Read Error.
	DRV_I2S_WRITE_ERROR	I2S Driver Write Error.

Structures

	Name	Description
	DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data
	DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	DRV_I2S_INTERFACE	This structure defines a structure of I2S Driver function pointers.

Types

	Name	Description
	DRV_I2S_BUFFER_EVENT_HANDLER	Pointer to a I2S Driver Buffer Event handler function
	DRV_I2S_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.

Description

I2S Driver Interface

The I2S device driver provides a simple interface to manage the I2S module on Microchip microcontrollers. This file provides the interface definition for the I2S driver.

File Name

drv_i2s.h

Company

Microchip Technology Inc.

drv_i2s_config_template.h

I2S Driver Configuration Template.

Macros

	Name	Description
	DRV_I2S_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_I2S_INDEX	I2S Static Index selection
	DRV_I2S_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2S_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2S_INTERRUPT_SOURCE_ERROR	Defines the interrupt source for the error interrupt
	DRV_I2S_INTERRUPT_SOURCE_RECEIVE	Macro to define the Receive interrupt source in case of static driver
	DRV_I2S_INTERRUPT_SOURCE_TRANSMIT	Macro to define the Transmit interrupt source in case of static driver
	DRV_I2S_PERIPHERAL_ID	Configures the I2S PLIB Module ID
	DRV_I2S_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL	Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver
	DRV_I2S_RECEIVE_DMA_CHANNEL	Macro to defines the I2S Driver Receive DMA Channel in case of static driver
	DRV_I2S_STOP_IN_IDLE	Identifies whether the driver should stop operations in stop in Idle mode.
	DRV_I2S_TRANSMIT_DMA_CHANNEL	Macro to defines the I2S Driver Transmit DMA Channel in case of static driver

Description

I2S Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_i2s_config_template.h

Company

Microchip Technology Inc.

Input Capture Driver Library

This section describes the Input Capture Driver Library.

Introduction

The Input Capture Static Driver provides a high-level interface to manage the Input Capture module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Starting/Stopping of the capture
- 16/32-bit data reads
- Buffer empty status

Library Interface

Functions

	Name	Description
	DRV_IC_Initialize	Initializes the Input Capture instance for the specified driver index. **Implementation:** Static
	DRV_IC_BufferIsEmpty	Returns the Input Capture instance buffer empty status for the specified driver index. **Implementation:** Static
	DRV_IC_Capture16BitDataRead	Reads the 16-bit Input Capture for the specified driver index. **Implementation:** Static

	DRV_IC_Capture32BitDataRead	Reads the 32-bit Input Capture for the specified driver index. **Implementation:** Static
	DRV_IC_Start	Starts the Input Capture instance for the specified driver index. **Implementation:** Static
	DRV_IC_Stop	Stops the Input Capture instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Input Capture Driver Library.

Functions

DRV_IC_Initialize Function

Initializes the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Initialize();
```

Returns

None.

Description

This routine initializes the Input Capture driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Input Capture module ID. For example, driver instance 0 can be assigned to Input Capture 2.

Remarks

This routine must be called before any other Input Capture routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_IC_Initialize(void)

DRV_IC_BufferIsEmpty Function

Returns the Input Capture instance buffer empty status for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
bool DRV_IC_BufferIsEmpty();
```

Returns

Boolean

- 1 - Buffer is empty
- 0 - Buffer is not empty

Description

Returns the Input Capture instance buffer empty status for the specified driver index. The function should be called to determine whether or not the IC buffer has data.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

bool DRV_IC_BufferIsEmpty(void)

DRV_IC_Capture16BitDataRead Function

Reads the 16-bit Input Capture for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
uint16_t DRV_IC_Capture16BitDataRead();
```

Returns

uint16_t value of the data read from the Input Capture.

Description

This routine reads the 16-bit data for the specified driver index.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

uint16_t DRV_IC_Capture16BitDataRead(void)

DRV_IC_Capture32BitDataRead Function

Reads the 32-bit Input Capture for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
uint32_t DRV_IC_Capture32BitDataRead();
```

Returns

uint32_t value of the data read from the Input Capture.

Description

This routine reads the 32-bit data for the specified driver index

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

uint32_t DRV_IC_Capture32BitDataRead(void)

DRV_IC_Start Function

Starts the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Start();
```

Returns

None.

Description

This routine starts the Input Capture driver for the specified driver index, starting an input capture.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

void DRV_IC_Start(void)

DRV_IC_Stop Function

Stops the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Stop();
```

Returns

None.

Description

This routine stops the Input Capture driver for the specified driver index, stopping an input capture.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

void DRV_IC_Stop(void)

Input System Service Touch Driver Library

This section describes the Touch Driver Libraries that support the Input System Service. These libraries are variants of libraries previously created to support the Touch System Service, which is being deprecated by the Input System Service.

Touch Driver Libraries in service of the Touch System Service:

ADC Touch Driver Library This topic describes the ADC Touch Driver Library.

mXT336T Touch Driver Library This topic describes the mXT336T Touch Driver Library.

Input System Service Touch ADC Driver Library

This touch driver library establishes a software resistive touch controller using the Analog-to-Digital Converter (ADC) module. This driver provides application routines to read touch input data from the touch screen. Touch events are detected using an interrupt service routine rather than polling. This driver also allows provides the capability to set translation coefficients that allow an application to map a raw screen values to actual native display size using through 4-point calibration technique.

Using the Library

This topic describes the basic architecture of the Touch ADC Driver Library for the Input System Service and provides information and examples on its use.

Interface Header File:

`<project>/firmware/src/system_config/<target_config>/driver/input/touch_adc/drv_touch_adc.h`

The interface to the Touch ADC Driver library is defined in the drv_touch_adc.h header file related to the currently active project target configuration. This file is automatically generated by MHGC. Any C language source (`.c`) file that uses the ADC Touch Driver library should include this header.

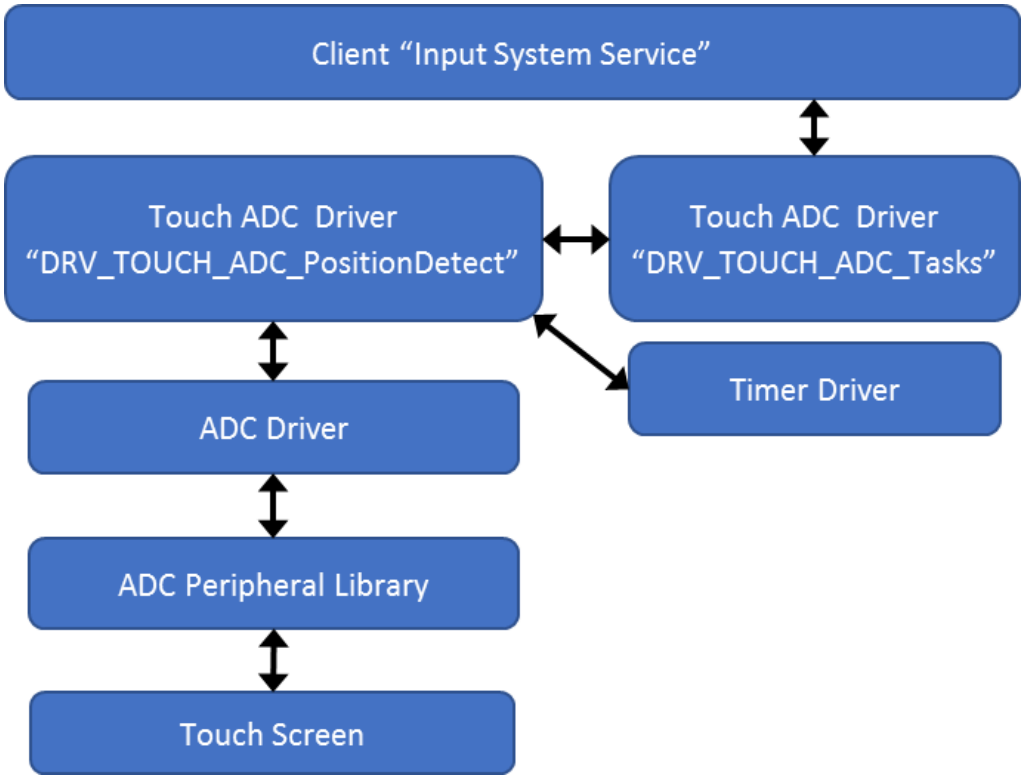
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This topic describes the components of the Touch ADC solution in support of the Input System Service.

- Touch ADC Driver – Finite-State machine that biases the x-axis and y-axis analog pins to measure resistance on a touch screen. Identifies touch events up, down or move. Forwards events to Harmony Service, Input System.
- ADC Driver – Provides the 4-wire resistive touch interface. It is used to driver hardware pin configuration to sample and measure touch screen resistance.
- Input system Service – Client level service which makes available touch events to the graphics library.

Touch ADC Driver Abstraction Model.



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Touch ADC Driver module.

Library Interface Section	Description
System Functions	Provides system interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, open, close, and status functions.
- Timer – rate at which to detect a new position

Of note are the following routine, which are found in:

`<project>/firmware/src/system_config/<target_config>/driver/input/touch_adc/src/drv_touch_adc.c`:

- `DRV_TOUCH_ADC_Tasks()` - sends a move, release, or press event to SYS_Input Services. It is called from `system_tasks.c`.
- `DRV_TOUCH_ADC_PositionDetect()` – performs resistance measurements to determine x and y positions.
- `DRV_TOUCH_ADC_TouchGetX()` – returns the x coordinate.
- `DRV_TOUCH_ADC_TouchGetY()` – returns the y coordinate.
- `DRV_TOUCH_ADCCoefficientSet()` – stores 4-translation calibrate coefficients

Initializing the Driver

Before the Touch ADC Driver can be opened, it must be configured and initialized. MHGC automatically includes the needed #define constants and source code into the project's `system_init.c` file.

The driver initialization is configured through the `DRV_TOUCH_ADC_INIT` data structure that is passed to the `DRV_TOUCH_ADC_Initialize` function.

Opening the Driver

To use the Touch ADC Driver, the application must open the driver. This is done by calling the `DRV_TOUCH_ADC_Open` function. If successful, the `DRV_TOUCH_ADC_Open` function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The `DRV_TOUCH_ADC_Open` function may return `DRV_HANDLE_INVALID` in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
DRV_HANDLE handle;
handle = DRV_TOUCH_ADC_Open( DRV_TOUCH_ADC_INDEX_0,DRV_IO_INTENT_EXCLUSIVE );
if( DRV_HANDLE_INVALID == handle )
{
// Unable
}
```

Tasks Routine

This routine communicates with the System Input Service on the state of the current X and Y positions detected. The routine checks 4 different conditions:

1. Invalid position – does not send an event upstream.
2. Same position – sends a still event to input system service
3. Move position – sends a move event to input system service.
4. Release condition – sends a release event to input system service
5. Press condition- sends a press event to input system service.

Touch Detection

This routine uses the services of the Touch ADC to establish a voltage divider on 4 ADC pins. The table below shows the pin configurations required to read the x and y values. The X value is read by a bias on x-axis and a measurement on ADC Y+ pin. The Y value is read by a bias on the y-axis and a measurement on ADC X+ pin.

Operation	X+ Pin Action	Y+Pin Action	X-GPIO Pin State	Y-GPIO Pin State
Get X	Pin Output (set)	Pin Input (Read X value)	Pin Input	Pin Output (clear)
Get Y	Pin Input (read Y value)	Pin Output (set)	Pin Output (clear)	Pin Input

(Here "set" means SYS_PORTS_PinSet() and "clear" means SYS_PORTS_PinClear().)

Configuring the Library

The configuration of the Touch ADC Driver is performed using the MPLAB Harmony Configurator.

Building the Library

The section lists the files that are available in the Touch ADC Library.

Description

The section lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/input/drv_touch_adc.h`

Interface File(s)

This table lists and describes the header files that must be included (i.e., using #include) by any code that uses this library. Note this file is automatically included in the project by MHGC.

Source File Name
`<project>/firmware/src/system_config/<target_config>/driver/input/touch_adc/drv_touch_adc.h`

Required File(s)

All of the required files listed below are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must always be included in the MPLAB X IDE project to build this library.

Source File Name
`<project>/firmware/src/system_config/<target_config>/driver/input/touch_adc/src/drv_touch_adc.c`

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The ADC Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Input System Service Library
- ADC Driver Library

Library Interface

Files

Files

Name	Description
drv_touch_adc.h	Touch ADC Driver interface file.

Description

drv_touch_adc.h

Touch ADC Driver interface file.

Description

Touch ADC Driver Interface File

This is a simple 4-wire resistive touch screen driver. The file consist of touch controller ADC driver interfaces. It implements the driver interfaces which read the touch input data from display overlay through the ADC peripheral.

Remarks

This driver is based on the MPLAB Harmony ADC driver.

File Name

drv_touch_adc.c

Input System Service mXT336T Touch Driver Library

This topic describes the mXT336T Touch Driver Library that supports theInput System Service.

The library provides an interface to manage the mXT336T Touch Driver module on the Microchip family of microcontrollers in different modes of operation. It supports the Input System Service as a client by providing touch events detected by the maXTouch® mXT336T Capacitive Touch Controller.

Description

The MPLAB Harmony mXT336T Touch Driver provides a high-level interface to the mXT336T Capacitive Touch Controller. This driver provides application routines to read the touch input data from the touch screen. Currently, the mXT336T Touch Driver supports non-gestural single-fingered and gestural two-finger touch inputs.

The mXT336T Capacitive Touch Controller notifies the host of the availability of touch input data through an external interrupt on the host. The mXT336T driver allows the application to map a controller pin as the external interrupt pin used by the mXT336T.

The driver contains the standard MPLAB Harmony driver interfaces including: initialization, destruction, status, tasks, open, close, and interrupt-driven read.

The driver contains no direct access to input events. All driver output is directed towards the MPLAB Harmony Input System Service and applications desiring to listen to input events must register with that service.

The aria_quickstart demonstration interfaces with the mXT336T Touch Driver Library. Please refer to the What is MPLAB Harmony? section in Volume I of MPLAB Harmony's built-in documentation for how the driver interacts with the framework.

Using the Library

This topic describes the basis architecture of the mXT336T Touch Driver Library that supports the Input System Service and provides information and examples on its use.

Description

Interface Header File: `./framework/driver/input/touch/mxt336t/drv_mxt336t.h`

The interface to the mXT336T Touch Driver library is defined in the `drv_mxt336t.h` header file. Any C language source (`.c`) file that uses the mXT336T Touch Driver library should include this header.

The mXT336T Touch Driver is based on the Object Protocol for the maXTouch® mXT336T Touchscreen Controller.

The aria_quickstart demonstration interfaces with the mXT336T Touch Driver Library. Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the mXT336T Touch Driver Library on the Microchip family microcontrollers. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

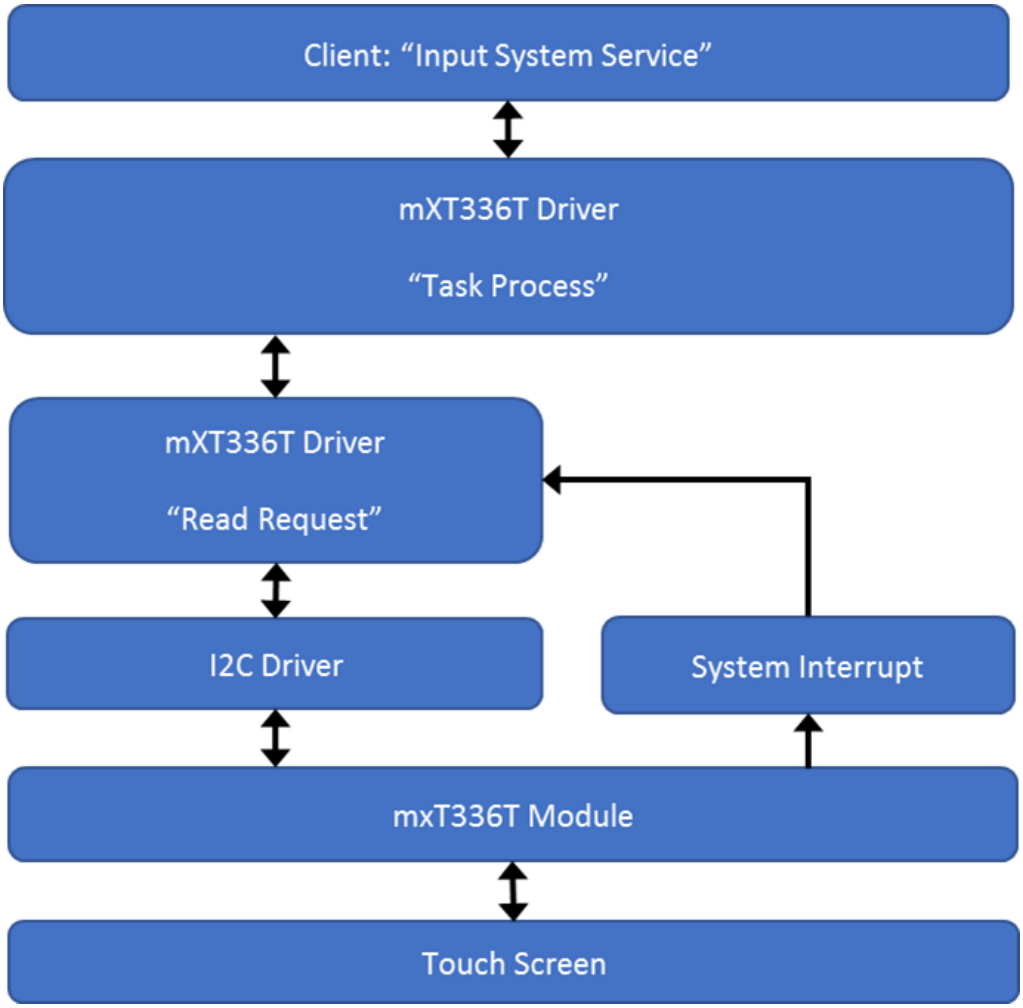
The mXT336T Touch Driver has routines to perform the following operations:

- Sending read request
- Reading the touch input data
- Access to the touch input data

The driver initialization routines allow the application to initialize the driver. The driver must be initialized before it can be used by application. Once

the touch input is available (by the assertion of the external interrupt input to the host) a touch input read request is sent to the mXT336t and input data is retrieved in a buffer. The buffer data is then decoded to get the x and y coordinate of the touch screen in the form of the number of pixels. After touch event data has been received it is propagated to the Input System Service which then distributes it to all interested parts of the application.

mXT336T Driver Abstraction Model



Library Overview

This section contains information about how the Touch Driver operates in a system.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the mXT336T Touch Driver.

Library Interface Section	Description
Device-Specific Functions	Provides mXT336T-specific system module interfaces, device initialization, deinitialization, open, close, task, and status functions.
Generic Functions	Provides generic system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

This section describes the workings of this Touch Driver library.

Description

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

- Read Request function, which provides Touch input data read request function.

Initializing the Driver

Before the mXT336T Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_MXT336T_INIT data structure that is passed to the DRV_MXT336T_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the mXT336T Touch Driver.

Example:

```
/* The following code shows an example of designing the
* DRV_TOUCH_INIT data structure. It also shows how an example
* usage of the DRV_TOUCH_MXT336T_Initialize function.
*/
This entire example section can be replaced with:

const DRV_MXT336T_INIT drvMXT336TInitData =
{
.drvOpen =  DRV_I2C_Open,
.orientation = 0,
.horizontalResolution = 480,
.verticalResolution = 272,
};

sysObj.drvMXT336T = DRV_MXT336T_Initialize(0, (SYS_MODULE_INIT *)&drvMXT336TInitData);
```

Touch Input Read Request

To read the touch input from the mXT336T touch controller device, a read request must be registered. This is done by calling DRV_MXT336T_ReadRequest. If successful, it registers a buffer read request to the I2C command queue. It also adds an input decode command to the mXT336T command queue once the I2C returns with touch input data. It can return error if the driver instance object is invalid or the mXT336T command queue is full. The read request is to be called from the mXT336T ISR. This ISR is triggered once the touch input is available.

The following code shows an example of a mXT336T read request registration:

```
SYS_MODULE_OBJ object; // Returned from DRV_TOUCH_MXT336T_Initialize
void ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMxt336t(void)
{
DRV__MXT336T_ReadRequest ( object );
// Do Other Tasks
.
.
.
}
```

Tasks Routine

This routine processes the mXT336T commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The mXT336T Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
SYS_MODULE_OBJ drvMXT336T;
SYS_MODULE_OBJ drvMxt0;; // Returned from DRV_TOUCH_MXT336T_Initialize
void SYS_Tasks( void )
{
DRV_MXT336T_Tasks(sysObj.drvMXT336T);
DRV_MXT_Tasks(sysObj.drvMxt0);
// Do other tasks
}
```

Configuring in MPLAB Harmony Configurator

The graphics demo aria_quickstart has several target configurations that provide examples of driver setup. Any configuration ending in meb2 or _meb2_wvga can be used.

The MPLAB Harmony Configurator Pin Settings tab should be configured to select the correct pin for the external interrupt. For example,

pic32mk_gp_db:

91	RF6	5V	MXT336T_TOUCH_INT	GPIO_CN

pic32mz_da_sk_extddr:

A14	RB1	-	MXT336T_TOUCH_INT	INT4

pic32mz_da_sk_intddr, pic32mz_da_noddr:

B9	RB1	-	MXT336T_TOUCH_INT	INT4

pic32mz_ef_sk:

23	RE8	-	MXT336T_TOUCH_INT	INT1

Library Interface

Files

Files

Name	Description
drv_input_mxt336t.h	Touch controller MXT336T Driver interface header file.

Description

drv_input_mxt336t.h

Touch controller MXT336T Driver interface header file.

Description

Touch Controller MXT336T Driver Interface File

This header file describes the macros, data structure and prototypes of the touch controller MXT336T driver interface.

File Name

drv_MXT336T.c

MIIM Driver Library

This section describes the MII Management (MIIM) Driver Library.

Introduction

The MIIM Driver library provides access to the MII Management interface (MIIM) of the Microchip PIC32 microcontrollers.

Description

The MIIM Driver is implemented as a driver object that provides APIs for:

- Asynchronous read/write and scan operations for accessing the external PHY registers
- Notification when MIIM operations have completed
- Driver status information
- Possibility to query or abort an ongoing operation.

Using the Library

This topic describes the basic architecture of the MIIM Driver Library and provides information and examples about its use.

Description

Interface Header File: `drv_miim.h`

The interface to the MIIM library is defined in the `drv_miim.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MIIM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

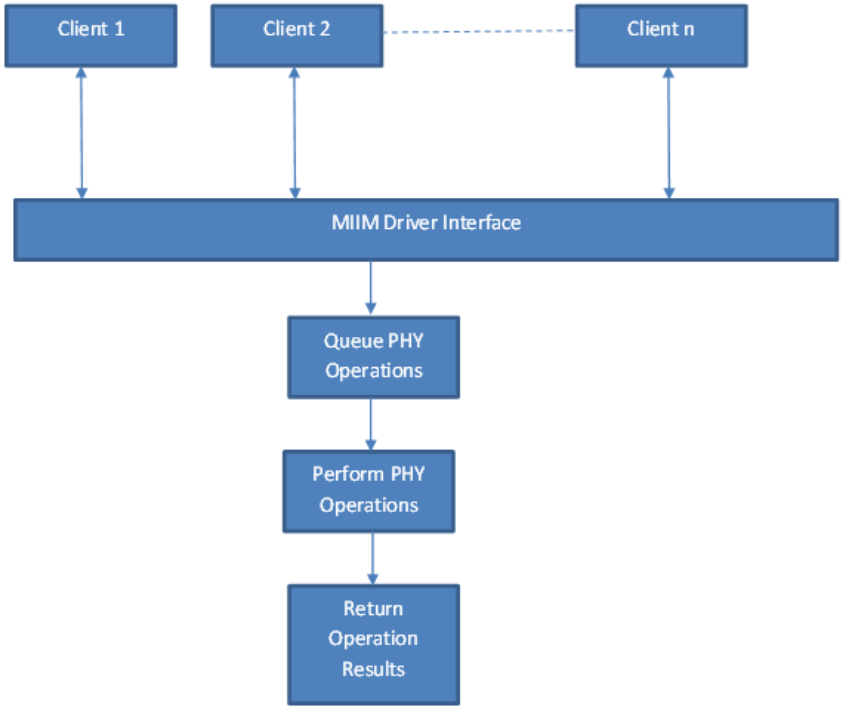
Description

The MIIM driver clients access PHY registers using the MIIM Driver API. The driver abstracts out the hardware details of the MIIM interface and provides a PHY register access mechanism to the application. The MIIM Driver provides read, write, and scan access to the PHY registers, together with driver and operation status APIs. The driver schedules operations requested by multiple clients and performs them sequentially, informing the clients about the operations outcome.

The user can poll for a certain operation status or can register callbacks to be notified of the completion of a scheduled operation.

A scheduled operation can be aborted, if not yet started.

MIIM Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information about how the driver operates in a system.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MIIM module.

Library Interface Section	Description
Functions	This section provides general interface routines.
Data Types and Constants	This section provides various definitions describing this API.

Configuring the Library

This section contains related configuration macros.

Macros

	Name	Description
	DRV_MIIM_INDEX_0	MIIM driver index definitions.
	DRV_MIIM_INDEX_COUNT	Number of valid MIIM driver indices.
	_DRV_MIIM_CONFIG_H	This is macro _DRV_MIIM_CONFIG_H.
	DRV_MIIM_CLIENT_OP_PROTECTION	Enables/Disables Client Operation Protection feature.
	DRV_MIIM_COMMANDS	Enables/Disables MIIM commands feature.
	DRV_MIIM_INSTANCE_CLIENTS	Selects the maximum number of clients.
	DRV_MIIM_INSTANCE_OPERATIONS	Selects the maximum number of simultaneous operations for an instance.
	DRV_MIIM_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

The configuration of the MIIM Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the MIIM Driver. Based on the selections made, the MIIM Driver may support the selected features. These configuration settings will apply to all instances of the MIIM Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_MIIM_INDEX_0 Macro

MIIM driver index definitions.

File

drv_miim.h

C

```
#define DRV_MIIM_INDEX_0 0
```

Description

MIIM Driver Module Index Numbers

These constants provide the MIIM driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_MIIM_Initialize and DRV_MIIM_Open routines to identify the driver instance in use.

DRV_MIIM_INDEX_COUNT Macro

Number of valid MIIM driver indices.

File

drv_miim.h

C

```
#define DRV_MIIM_INDEX_COUNT 1
```

Description

MIIM Driver Module Index Count

This constant identifies the number of valid MIIM driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

_DRV_MIIM_CONFIG_H Macro

File

drv_miim_config.h

C

```
#define _DRV_MIIM_CONFIG_H
```

Description

This is macro _DRV_MIIM_CONFIG_H.

DRV_MIIM_CLIENT_OP_PROTECTION Macro

Enables/Disables Client Operation Protection feature.

File

drv_miim_config.h

C

```
#define DRV_MIIM_CLIENT_OP_PROTECTION 0
```

Description

MIIM client Operation Protection

Because of the recirculation of the operation handles and client handles the possibility exists that a misbehaved client inadvertently gets the results of a previous completed operations that now belongs to a different client. When this feature is enabled, extra protection is added for an operation handle to uniquely identify a client that has started the operation and extra check is done that operation belongs to the client that asks for the result.

Remarks

Set the value to 1 to enable, 0 to disable the feature.

Enabling this feature requires a small overhead in code and data size.

DRV_MIIM_COMMANDS Macro

Enables/Disables MIIM commands feature.

File

drv_miim_config.h

C

```
#define DRV_MIIM_COMMANDS 0
```

Description

MIIM PHY Commands

Adds a MIIM command to the TCP/IP command menu allowing to read/write a PHY register.

Remarks

Set the value to 1 to enable, 0 to disable the feature.

Currently the MIIM commands are integrated in the TCP/IP commands. To have the MIIM commands available the TCP/IP commands need to be enabled.

DRV_MIIM_INSTANCE_CLIENTS Macro

Selects the maximum number of clients.

File

drv_miim_config.h

C

```
#define DRV_MIIM_INSTANCE_CLIENTS 2
```

Description

MIIM number of clients

This definition select the MIIM Maximum Number of Clients per instance.

Remarks

By default the 1st MIIM client is the DRV_ETHPHY. An extra client is allowed.

DRV_MIIM_INSTANCE_OPERATIONS Macro

Selects the maximum number of simultaneous operations for an instance.

File

drv_miim_config.h

C

```
#define DRV_MIIM_INSTANCE_OPERATIONS 4
```

Description

MIIM instance operations

This definition selects the maximum number of simultaneous operations that can be supported by this driver. Note that this represents operations for all clients

Remarks

None.

DRV_MIIM_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_miim_config.h

C

```
#define DRV_MIIM_INSTANCES_NUMBER 1
```

Description

MIIM hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Usually set to 1.

Remarks

None.

Building the Library

This section lists the files that are available in the MIIM Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/miim/`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_miim.h`	This is the MIIM Driver Library's interface header file.
`/config/drv_miim.config.h`	This file contains the configuration macros.

Required File(s)

MHC

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_miim.c`	This file contains the source code for the dynamic implementation of the MIIM Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Library Interface

This section describes the Application Programming Interface (API) functions of the MIIM Driver Library.

Refer to each section for a detailed description.

a) Functions

	Name	Description
	DRV_MIIM_ClientStatus	Gets the current client-specific status the MIIM driver.
	DRV_MIIM_Close	Closes an opened instance of the MIIM driver.
	DRV_MIIM_Deinitialize	Deinitializes the specified instance of the MIIM driver module.
	DRV_MIIM_DeregisterCallback	Deregisters an notification callback function for the client operations.
	DRV_MIIM_Initialize	Initializes the MIIM driver.
	DRV_MIIM_Open	Opens the specified MIIM driver instance and returns a handle to it.
	DRV_MIIM_OperationAbort	Aborts a current client operation initiated by the MIIM driver.
	DRV_MIIM_OperationResult	Gets the result of a client operation initiated by the MIIM driver.
	DRV_MIIM_Read	Initiates a SMI/MIIM read transaction.
	DRV_MIIM_RegisterCallback	Registers an notification callback function for the client operations.
	DRV_MIIM_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_MIIM_Scan	Initiates a SMI/MIIM scan (periodic read)transaction.
	DRV_MIIM_Setup	Sets up a MIIM client.
	DRV_MIIM_Status	Provides the current status of the MIIM driver module.
	DRV_MIIM_Tasks	Maintains the driver's state machine.
	DRV_MIIM_Write	Initiates a SMI/MIIM write transaction.

b) Data Types and Constants

	Name	Description
	DRV_MIIM_INIT	Contains all the data necessary to initialize the MIIM device.
	DRV_MIIM_OBJECT_BASE	Declaration of a MIIM base object.
	DRV_MIIM_CALLBACK_HANDLE	Handle that identifies a client registration operation.
	DRV_MIIM_CLIENT_STATUS	Defines the possible results of operations that can succeed or fail
	DRV_MIIM_OPERATION_CALLBACK	Notification function that will be called when a MIIM operation is completed and the driver client needs to be notified.
	DRV_MIIM_OPERATION_FLAGS	List of flags that apply to a client operation.
	DRV_MIIM_OPERATION_HANDLE	MIIM operation handle.
	DRV_MIIM_SETUP	Contains all the data necessary to set up the MIIM device.
	DRV_MIIM_SETUP_FLAGS	List of flags that apply to a client setup operation.
	DRV_MIIM_OBJECT_BASE_Default	The supported basic MIIM driver (DRV_MIIM_OBJECT_BASE). This object is implemented by default as using the standard MIIM interface. It can be overwritten dynamically when needed.

a) Functions

DRV_MIIM_ClientStatus Function

Gets the current client-specific status the MIIM driver.

File

drv_miim.h

C

```
DRV_MIIM_CLIENT_STATUS DRV_MIIM_ClientStatus(DRV_HANDLE handle);
```

Returns

- DRV_MIIM_CLIENT_STATUS_READY - if the client handle represents a valid MIIM client
- DRV_MIIM_CLIENT_STATUS_ERROR - if the client handle is an invalid MIIM client

Description

This function gets the client-specific status of the MIIM driver associated with the given handle.

Remarks

This function can be used to check that a client handle points to a valid MIIM client. The MIIM driver queues operations so it will always return DRV_MIIM_CLIENT_STATUS_READY.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_CLIENT_STATUS DRV_MIIM_ClientStatus(DRV_HANDLE handle)

DRV_MIIM_Close Function

Closes an opened instance of the MIIM driver.

File

drv_miim.h

C

```
void DRV_MIIM_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MIIM driver, invalidating the handle.

Remarks

- After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_MIIM_Open before the caller may use the driver again.
- Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_MIIM_Initialize routine must have been called for the specified MIIM driver instance.

DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_MIIM_Open
```

```
DRV_MIIM_Close(handle);
```

Function

void DRV_MIIM_Close(DRV_HANDLE handle)

DRV_MIIM_Deinitialize Function

Deinitializes the specified instance of the MIIM driver module.

File

drv_miim.h

C

```
void DRV_MIIM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the MIIM driver module, disabling its operation (and any hardware)and invalidates all of the internal data.

Remarks

- Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The DRV_MIIM_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

Function

void DRV_MIIM_Deinitialize (SYS_MODULE_OBJ object)

DRV_MIIM_DeregisterCallback Function

Deregisters an notification callback function for the client operations.

File

drv_miim.h

C

```
DRV_MIIM_RESULT DRV_MIIM_DeregisterCallback(DRV_HANDLE handle, DRV_MIIM_CALLBACK_HANDLE cbHandle);
```

Returns

- DRV_MIIM_RES_OK if the operation succeeded.
- an error code otherwise

Description

This function deregisters a previously registered client notification callback function.

Remarks

There is only one notification callback function available per client. To register a new callback function use DRV_MIIM_DeregisterCallback first.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_RESULT DRV_MIIM_DeregisterCallback(DRV_HANDLE handle, DRV_MIIM_CALLBACK_HANDLE cbHandle);

DRV_MIIM_Initialize Function

Initializes the MIIM driver.

File

drv_miim.h

C

```
SYS_MODULE_OBJ DRV_MIIM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- a valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

This function initializes the MIIM driver, making it ready for clients to open and use it.

Remarks

- This function must be called before any other MIIM routine is called.
- This function should only be called once during system initialization unless DRV_MIIM_Deinitialize is called to deinitialize the driver instance.
- The returned object must be passed as argument to DRV_MIIM_Reinitialize, DRV_MIIM_Deinitialize, DRV_MIIM_Tasks and DRV_MIIM_Status routines.

Preconditions

None.

Example

Function

SYS_MODULE_OBJ DRV_MIIM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init)

DRV_MIIM_Open Function

Opens the specified MIIM driver instance and returns a handle to it.

File

drv_miim.h

C

```
DRV_HANDLE DRV_MIIM_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- valid open-instance handle if successful (a number identifying both the caller and the module instance).
- DRV_HANDLE_INVALID if an error occurs

Description

This function opens the specified MIIM driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_MIIM_Close routine is called.

This function will NEVER block waiting for hardware.

Preconditions

The DRV_MIIM_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_MIIM_Open(DRV_MIIM_INDEX_0, 0);
if (DRV_HANDLE_INVALID == handle)
```

```
{
    // Unable to open the driver
}
```

Function

DRV_HANDLE DRV_MIIM_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent)

DRV_MIIM_OperationAbort Function

Aborts a current client operation initiated by the MIIM driver.

File

drv_miim.h

C

```
DRV_MIIM_RESULT DRV_MIIM_OperationAbort(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle);
```

Returns

DRV_MIIM_RES_OK for success; operation has been aborted

< 0 - an error has occurred and the operation could not be completed

Description

Aborts a current client operation initiated by the MIIM driver.

Remarks

This operation will stop/abort a scan operation started by DRV_MIIM_Scan.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.
- A driver operation was started

Example

Function

DRV_MIIM_RESULT DRV_MIIM_OperationAbort(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle)

DRV_MIIM_OperationResult Function

Gets the result of a client operation initiated by the MIIM driver.

File

drv_miim.h

C

```
DRV_MIIM_RESULT DRV_MIIM_OperationResult(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle, uint16_t*
pOpData);
```

Returns

- DRV_MIIM_RESULT value describing the current operation result: DRV_MIIM_RES_OK for success; operation has been completed successfully and pOpData updated DRV_MIIM_RES_PENDING operation is in progress an DRV_MIIM_RESULT error code if the operation failed.

Description

Returns the result of a client operation initiated by the MIIM driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

This function returns the result of the last driver operation. It will return DRV_MIIM_RES_PENDING if an operation is still in progress. Otherwise a DRV_MIIM_RESULT describing the operation outcome.

Note that for a scan operation DRV_MIIM_RES_PENDING will be returned when there's no new scan data available. DRV_MIIM_RES_OK means the scan data is fresh.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.
- A driver operation was started

Example

Function

DRV_MIIM_RESULT DRV_MIIM_OperationResult(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle, uint16_t* pOpData)

DRV_MIIM_Read Function

Initiates a SMI/MIIM read transaction.

File

drv_miim.h

C

```
DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Read(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd,
DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
```

Returns

A not NULL DRV_MIIM_OPERATION_HANDLE if the operation was successfully scheduled. NULL if the operation failed. More details in pOpResult.

Description

This function initiates a SMI/MIIM read transaction for a given MIIM register.

Remarks

If operation was scheduled successfully, the result will be DRV_MIIM_RES_PENDING. Otherwise an error code will be returned.

Upon the operation completion:

- If the operation is to be discarded (DRV_MIIM_OPERATION_FLAG_DISCARD is set) there will be no notification to the client. The operation associated resources will be released.
- If the operation is not to be discarded, then:
- if the client has registered an operation notification callback (DRV_MIIM_RegisterCallback) then the callback will be called. After that the operation associated resources will be released.
- if there is no notification callback the MIIM driver will wait for the client to poll and read the operation result using DRV_MIIM_OperationResult(). Only then the operation will be released.

A completed non-discardable operation will remain available for returning the result until the client is somehow notified of the operation result. When polling is used, DRV_MIIM_OperationResult() needs to be called to free the operation associated resources.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Read(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd, DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);

DRV_MIIM_RegisterCallback Function

Registers an notification callback function for the client operations.

File

drv_miim.h

C

```
DRV_MIIM_CALLBACK_HANDLE DRV_MIIM_RegisterCallback(DRV_HANDLE handle, DRV_MIIM_OPERATION_CALLBACK
```

```
cbFunction, DRV_MIIM_RESULT* pRegResult);
```

Returns

- a valid DRV_MIIM_CALLBACK_HANDLE if the operation succeeded.
- NULL otherwise

Description

This function registers a client callback function. The function will be called by the MIIM driver when a scheduled operation is completed.

Remarks

There is only one notification callback function available per client. To register a new callback function use DRV_MIIM_DeregisterCallback first.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_CALLBACK_HANDLE DRV_MIIM_RegisterCallback(DRV_HANDLE handle, DRV_MIIM_OPERATION_CALLBACK cbFunction, DRV_MIIM_RESULT* pRegResult);

DRV_MIIM_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

File

drv_miim.h

C

```
void DRV_MIIM_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Remarks

- This function can be called multiple times to reinitialize the module.
- This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.
- This function is currently NOT IMPLEMENTED.

Preconditions

The DRV_MIIM_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

Function

void DRV_MIIM_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init)

DRV_MIIM_Scan Function

Initiates a SMI/MIIM scan (periodic read)transaction.

File

drv_miim.h

C

```
DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Scan(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd,
```

```
DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
```

Returns

A not NULL DRV_MIIM_OPERATION_HANDLE if the operation was successfully scheduled. NULL if the operation failed. More details in pOpResult.

Description

This function initiates a SMI/MIIM scan transaction for a given MIIM register.

Remarks

If operation was scheduled successfully, the result will be DRV_MIIM_RES_PENDING. Otherwise an error code will be returned.

When a new scan result is available:

- If the operation is to be discarded (DRV_MIIM_OPERATION_FLAG_DISCARD is set) there will be no notification to the client.
- If the operation is not to be discarded, then:
- if the client has registered an operation notification callback (DRV_MIIM_RegisterCallback) then the notification callback will be called.
- if there is no notification callback the MIIM driver will wait for the client to poll and read the operation result using DRV_MIIM_OperationResult(). Only then the operation will be released.

A scheduled scan operation will remain active in the background and will be available for returning the scan results. When polling is used, DRV_MIIM_OperationResult()will return the latest scan result. The operation associated resources will be released and scan stopped only when DRV_MIIM_OperationAbort() is called.

While scan is active all other transactions (including from other clients) will be inhibited! Use carefully!

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Scan(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd, DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);

DRV_MIIM_Setup Function

Sets up a MIIM client.

File

drv_miim.h

C

```
DRV_MIIM_RESULT DRV_MIIM_Setup(DRV_HANDLE handle, const DRV_MIIM_SETUP* pSetUp);
```

Returns

- DRV_MIIM_RES_OK if the setup operation has been performed successfully
- an DRV_MIIM_RESULT error code if the set up procedure failed.

Description

This function performs the set up of a MIIM client. It programs the MIIM operation using the supplied frequencies.

Remarks

None.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid device handle.

Example

Function

DRV_MIIM_RESULT DRV_MIIM_Setup(DRV_HANDLE handle, const DRV_MIIM_SETUP* pSetUp)

DRV_MIIM_Status Function

Provides the current status of the MIIM driver module.

File

drv_miim.h

C

```
SYS_STATUS DRV_MIIM_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the MIIM driver module.

Remarks

- Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.
- SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another
- SYS_STATUS_ERROR - Indicates that the driver is in an error state
- Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized
- If the status operation returns SYS_STATUS_BUSY, the a previous system level operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.
- The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.
- This function will NEVER block waiting for hardware.
- If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_MIIM_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_MIIM_Initialize
SYS_STATUS          status;

status = DRV_MIIM_Status(object);
if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_MIIM_Initialize

Function

SYS_STATUS DRV_MIIM_Status (SYS_MODULE_OBJ object)

DRV_MIIM_Tasks Function

Maintains the driver's state machine.

File

drv_miim.h

C

```
void DRV_MIIM_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine.

Remarks

- This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)
- This function will never block or access any resources that may cause it to block.

Preconditions

The DRV_MIIM_Initialize routine must have been called for the specified MIIM driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_MIIM_Initialize

while (true)
{
    DRV_MIIM_Tasks (object);

    // Do other tasks
}
```

Function

void DRV_MIIM_Tasks(SYS_MODULE_OBJ object)

DRV_MIIM_Write Function

Initiates a SMI/MIIM write transaction.

File

drv_miim.h

C

```
DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Write(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd, uint16_t
wData, DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
```

Returns

A not NULL DRV_MIIM_OPERATION_HANDLE if the operation was successfully scheduled. NULL if the operation failed. More details in pOpResult.

Description

This function initiates a SMI/MIIM write transaction for a given MIIM register.

Remarks

If operation was scheduled successfully, the result will be DRV_MIIM_RES_PENDING. Otherwise an error code will be returned.

Upon the operation completion:

- If the operation is to be discarded (DRV_MIIM_OPERATION_FLAG_DISCARD is set) there will be no notification to the client. The operation associated resources will be released.
- If the operation is not to be discarded, then:
- if the client has registered an operation notification callback (DRV_MIIM_RegisterCallback) then the notification callback will be called. After that the operation associated resources will be released.
- if there is no notification callback the MIIM driver will wait for the client to poll and read the operation result using DRV_MIIM_OperationResult(). Only then the operation will be released.

A completed non-discardable operation will remain available for returning the result until the client is somehow notified of the operation result. When polling is used, DRV_MIIM_OperationResult() needs to be called to free the operation associated resources.

A write operation normally uses DRV_MIIM_OPERATION_FLAG_DISCARD if it is not interested when the operation has completed.

Preconditions

- The DRV_MIIM_Initialize routine must have been called.
- DRV_MIIM_Open must have been called to obtain a valid opened device handle.

Example

Function

DRV_MIIM_OPERATION_HANDLE DRV_MIIM_Write(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd, uint16_t wData, DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);

b) Data Types and Constants

DRV_MIIM_INIT Structure

Contains all the data necessary to initialize the MIIM device.

File

drv_miim.h

C

```
struct DRV_MIIM_INIT {
  SYS_MODULE_INIT moduleInit;
  uintptr_t ethphyId;
};
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
uintptr_t ethphyId;	Identifies peripheral (PLIB-level) ID

Description

MIIM Device Driver Initialization Data

This data structure contains all the data necessary to initialize the MIIM device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_MIIM_Initialize routine.

DRV_MIIM_OBJECT_BASE Structure

Declaration of a MIIM base object.

File

drv_miim.h

C

```
struct DRV_MIIM_OBJECT_BASE {
  SYS_MODULE_OBJ (* DRV_MIIM_Initialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
  void (* DRV_MIIM_Reinitialize)(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
  void (* DRV_MIIM_Deinitialize)(SYS_MODULE_OBJ object);
  SYS_STATUS (* DRV_MIIM_Status)(SYS_MODULE_OBJ object);
  void (* DRV_MIIM_Tasks)(SYS_MODULE_OBJ object);
  DRV_HANDLE (* DRV_MIIM_Open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
  DRV_MIIM_RESULT (* DRV_MIIM_Setup)(DRV_HANDLE handle, const DRV_MIIM_SETUP* pSetUp);
  void (* DRV_MIIM_Close)(DRV_HANDLE handle);
  DRV_MIIM_CLIENT_STATUS (* DRV_MIIM_ClientStatus)(DRV_HANDLE handle);
  DRV_MIIM_CALLBACK_HANDLE (* DRV_MIIM_RegisterCallback)(DRV_HANDLE handle, DRV_MIIM_OPERATION_CALLBACK
cbFunction, DRV_MIIM_RESULT* pRegResult);
  DRV_MIIM_RESULT (* DRV_MIIM_DeregisterCallback)(DRV_HANDLE handle, DRV_MIIM_CALLBACK_HANDLE cbHandle);
  DRV_MIIM_OPERATION_HANDLE (* DRV_MIIM_Read)(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd,
DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
  DRV_MIIM_OPERATION_HANDLE (* DRV_MIIM_Write)(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd,
uint16_t wData, DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
  DRV_MIIM_OPERATION_HANDLE (* DRV_MIIM_Scan)(DRV_HANDLE handle, unsigned int rIx, unsigned int phyAdd,
DRV_MIIM_OPERATION_FLAGS opFlags, DRV_MIIM_RESULT* pOpResult);
  DRV_MIIM_RESULT (* DRV_MIIM_OperationResult)(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle,
uint16_t* pOpData);
```

```
  DRV_MIIM_RESULT (* DRV_MIIM_OperationAbort)(DRV_HANDLE handle, DRV_MIIM_OPERATION_HANDLE opHandle);
};
```

Description

MIIM Driver Base Object

This data structure identifies the required basic interface of the MIIM driver. Any dynamic MIIM driver has to export this interface.

Remarks

This object provides the basic MIIM functionality. Any derived driver can override the basic functionality while maintaining the required interface.

DRV_MIIM_CALLBACK_HANDLE Type

Handle that identifies a client registration operation.

File

drv_miim.h

C

```
typedef const void* DRV_MIIM_CALLBACK_HANDLE;
```

Description

Type: MIIM Callback Registration handle

A handle that a client obtains when calling DRV_MIIM_RegisterCallback. It can be used to deregister the notification callback: DRV_MIIM_DeregisterCallback

Remarks

A valid registration handle is not NULL. An invalid registration handle == 0.

DRV_MIIM_CLIENT_STATUS Enumeration

Defines the possible results of operations that can succeed or fail

File

drv_miim.h

C

```
typedef enum {
  DRV_MIIM_CLIENT_STATUS_ERROR,
  DRV_MIIM_CLIENT_STATUS_READY
} DRV_MIIM_CLIENT_STATUS;
```

Members

Members	Description
DRV_MIIM_CLIENT_STATUS_ERROR	Unspecified error condition. Client does not exist
DRV_MIIM_CLIENT_STATUS_READY	Up and running, can accept operations

Description

MIIM Driver Operation Result *

MIIM Driver Operation Result Codes

This enumeration defines the possible results of any of the MIIM driver operations that have the possibility of failing. This result should be checked to ensure that the operation achieved the desired result.

DRV_MIIM_OPERATION_CALLBACK Type

Notification function that will be called when a MIIM operation is completed and the driver client needs to be notified.

File

drv_miim.h

C

```
typedef void (* DRV_MIIM_OPERATION_CALLBACK)(DRV_HANDLE cliHandle, DRV_MIIM_OPERATION_HANDLE opHandle,
DRV_MIIM_RESULT opResult, uint16_t opData);
```

Description

Type: MIIM Driver Operation Complete Callback

The format of an operation callback notification function registered with the MIIM driver.

Remarks

None.

Parameters

Parameters	Description
cliHandle	the client handle. This is the handle that identifies the client (obtained with DRV_MIIM_Open) that initiated the operation.
opHandle	the operation handle. This is the handle that identifies the operation (obtained with DRV_MIIM_Read, DRV_MIIM_Write, etc.)
opResult	operation result DRV_MIIM_RES_OK if operation completed successfully, otherwise an error code
opData	operation specific data, only if the result is DRV_MIIM_RES_OK For read/scan operation this is the MIIM read data. For write operation this is that data that was written with MIIM.

DRV_MIIM_OPERATION_FLAGS Enumeration

List of flags that apply to a client operation.

File

drv_miim.h

C

```
typedef enum {
  DRV_MIIM_OPERATION_FLAG_NONE,
  DRV_MIIM_OPERATION_FLAG_DISCARD
} DRV_MIIM_OPERATION_FLAGS;
```

Members

Members	Description
DRV_MIIM_OPERATION_FLAG_NONE	No flag specified
DRV_MIIM_OPERATION_FLAG_DISCARD	Upon completion discard the operation result. The client will not poll to check the result nor will need notification This allows dummy operations, discarded as they complete

Description

MIIM Driver Operation flags

This enumeration identifies the operation-specific flags supported by the MIIM driver.

Remarks

Currently only 8 bit flags are supported.

Multiple flags can be simultaneously set.

DRV_MIIM_OPERATION_HANDLE Type

MIIM operation handle.

File

drv_miim.h

C

```
typedef const void* DRV_MIIM_OPERATION_HANDLE;
```

Description

Type: DRV_MIIM_OPERATION_HANDLE

A handle that identifies an operation started by a client. This handle can be used by the client to query the operation status, result, etc. It is also used when the operation complete notification occurs.

Remarks

A valid operation handle is not NULL. An invalid operation handle == 0.

DRV_MIIM_SETUP Structure

Contains all the data necessary to set up the MIIM device.

File

drv_miim.h

C

```
typedef struct {
  uint32_t hostClockFreq;
  uint32_t maxBusFreq;
  DRV_MIIM_SETUP_FLAGS setupFlags;
} DRV_MIIM_SETUP;
```

Members

Members	Description
uint32_t hostClockFreq;	The clock frequency on which this MIIM module operates on, Hz
uint32_t maxBusFreq;	The MIIM bus maximum supported frequency, Hz This is a maximum value. The actual generated value may differ.
DRV_MIIM_SETUP_FLAGS setupFlags;	Setup flags

Description

MIIM Device Driver Set up Data

This data structure contains all the data necessary to configure the MIIM device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_MIIM_Setup routine.

DRV_MIIM_SETUP_FLAGS Enumeration

List of flags that apply to a client setup operation.

File

drv_miim.h

C

```
typedef enum {
  DRV_MIIM_SETUP_FLAG_NONE,
  DRV_MIIM_SETUP_FLAG_PREAMBLE_SUPPRESSED,
  DRV_MIIM_SETUP_FLAG_PREAMBLE_DEFAULT,
  DRV_MIIM_SETUP_FLAG_SCAN_ADDRESS_INCREMENT,
  DRV_MIIM_SETUP_FLAG_SCAN_ADDRESS_DEFAULT
} DRV_MIIM_SETUP_FLAGS;
```

Members

Members	Description
DRV_MIIM_SETUP_FLAG_NONE	No flag specified. Default value
DRV_MIIM_SETUP_FLAG_PREAMBLE_SUPPRESSED	Suppress the normal 32 bit MIIM preamble field. Some PHYs support suppressed preamble
DRV_MIIM_SETUP_FLAG_PREAMBLE_DEFAULT	Include the 32 bit MIIM preamble field. Default operation.
DRV_MIIM_SETUP_FLAG_SCAN_ADDRESS_INCREMENT	Scan operation will read across a range of PHY addresses Scan will start with address 1 through the address set in the scan operation
DRV_MIIM_SETUP_FLAG_SCAN_ADDRESS_DEFAULT	Scan operation will read just one PHY address. Default operation.

Description

MIIM Driver Set up flags

This enumeration identifies the setup specific flags supported by the MIIM driver.

Remarks

Multiple flags can be simultaneously set.

DRV_MIIM_OBJECT_BASE_Default Variable

File

drv_miim.h

C

```
const DRV_MIIM_OBJECT_BASE DRV_MIIM_OBJECT_BASE_Default;
```

Description

The supported basic MIIM driver (DRV_MIIM_OBJECT_BASE). This object is implemented by default as using the standard MIIM interface. It can be overwritten dynamically when needed.

Files

Files

Name	Description
drv_miim.h	MIIM Device Driver Interface File
drv_miim_config.h	MIIM driver configuration definitions template.

Description

This section lists the source and header files used by the Media Interface Independent Management (MIIM)Driver Library.

drv_miim.h

MIIM Device Driver Interface File

Enumerations

	Name	Description
	DRV_MIIM_CLIENT_STATUS	Defines the possible results of operations that can succeed or fail
	DRV_MIIM_OPERATION_FLAGS	List of flags that apply to a client operation.
	DRV_MIIM_SETUP_FLAGS	List of flags that apply to a client setup operation.

Functions

	Name	Description
	DRV_MIIM_ClientStatus	Gets the current client-specific status the MIIM driver.
	DRV_MIIM_Close	Closes an opened instance of the MIIM driver.
	DRV_MIIM_Deinitialize	Deinitializes the specified instance of the MIIM driver module.
	DRV_MIIM_DeregisterCallback	Deregisters an notification callback function for the client operations.
	DRV_MIIM_Initialize	Initializes the MIIM driver.
	DRV_MIIM_Open	Opens the specified MIIM driver instance and returns a handle to it.
	DRV_MIIM_OperationAbort	Aborts a current client operation initiated by the MIIM driver.
	DRV_MIIM_OperationResult	Gets the result of a client operation initiated by the MIIM driver.
	DRV_MIIM_Read	Initiates a SMI/MIIM read transaction.
	DRV_MIIM_RegisterCallback	Registers an notification callback function for the client operations.
	DRV_MIIM_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_MIIM_Scan	Initiates a SMI/MIIM scan (periodic read)transaction.
	DRV_MIIM_Setup	Sets up a MIIM client.
	DRV_MIIM_Status	Provides the current status of the MIIM driver module.
	DRV_MIIM_Tasks	Maintains the driver's state machine.
	DRV_MIIM_Write	Initiates a SMI/MIIM write transaction.

Macros

	Name	Description
	DRV_MIIM_INDEX_0	MIIM driver index definitions.
	DRV_MIIM_INDEX_COUNT	Number of valid MIIM driver indices.

Structures

	Name	Description
	DRV_MIIM_INIT	Contains all the data necessary to initialize the MIIM device.
	DRV_MIIM_OBJECT_BASE	Declaration of a MIIM base object.
	DRV_MIIM_SETUP	Contains all the data necessary to set up the MIIM device.

Types

	Name	Description
	DRV_MIIM_CALLBACK_HANDLE	Handle that identifies a client registration operation.
	DRV_MIIM_OPERATION_CALLBACK	Notification function that will be called when a MIIM operation is completed and the driver client needs to be notified.
	DRV_MIIM_OPERATION_HANDLE	MIIM operation handle.

Variables

	Name	Description
	DRV_MIIM_OBJECT_BASE_Default	The supported basic MIIM driver (DRV_MIIM_OBJECT_BASE). This object is implemented by default as using the standard MIIM interface. It can be overwritten dynamically when needed.

Description

MIIM Device Driver Interface

The MIIM device driver provides a simple interface to manage an MIIM peripheral using MIIM (SMI)interface. This file defines the interface definitions and prototypes for the MIIM driver.

File Name

drv_miim.h

Company

Microchip Technology Inc.

drv_miim_config.h

MIIM driver configuration definitions template.

Macros

	Name	Description
	_DRV_MIIM_CONFIG_H	This is macro _DRV_MIIM_CONFIG_H.
	DRV_MIIM_CLIENT_OP_PROTECTION	Enables/Disables Client Operation Protection feature.
	DRV_MIIM_COMMANDS	Enables/Disables MIIM commands feature.
	DRV_MIIM_INSTANCE_CLIENTS	Selects the maximum number of clients.
	DRV_MIIM_INSTANCE_OPERATIONS	Selects the maximum number of simultaneous operations for an instance.
	DRV_MIIM_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

MIIM Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_miim_config.h

Company

Microchip Technology Inc.

Motor Control PWM (MCPWM) Driver Library

This section describes the MCPWM Driver Library.

Introduction

The MCPWM Static Driver provides a high-level interface to manage the MCPWM module on the Microchip family of microcontrollers.

Description

Through MHC, this driver provides APIs to initialize, enable, and disable the MCPWM module.

Library Interface

Function(s)

	Name	Description
	DRV_MCPWM_Disable	Disables the MCPWM instance for the specified driver index. **Implementation:** Static
	DRV_MCPWM_Enable	Enables the MCPWM instance for the specified driver index. **Implementation:** Static
	DRV_MCPWM_Initialize	Initializes the MCPWM instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the MCPWM Driver Library.

Function(s)

DRV_MCPWM_Disable Function

Disables the MCPWM instance for the specified driver index.

Implementation: Static

File

drv_mcpwm.h

C

```
void DRV_MCPWM_Disable();
```

Returns

None.

Description

This routine disables the MCPWM Driver instance for the specified driver instance.

Preconditions

DRV_MCPWM_Initialize has been called.

Function

void DRV_MCPWM_Disable(void)

DRV_MCPWM_Enable Function

Enables the MCPWM instance for the specified driver index.

Implementation: Static

File

drv_mcpwm.h

C

```
void DRV_MCPWM_Enable();
```

Returns

None.

Description

This routine enables the MCPWM Driver instance for the specified driver instance, making it ready for clients to use it. The enable routine is specified by the MHC parameters.

Preconditions

DRV_MCPWM_Initialize has been called.

Function

void DRV_MCPWM_Enable(void)

DRV_MCPWM_Initialize Function

Initializes the MCPWM instance for the specified driver index.

Implementation: Static

File

drv_mcpwm.h

C

```
void DRV_MCPWM_Initialize();
```

Returns

None.

Description

This routine initializes the MCPWM Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This routine must be called before any other MCPWM routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_MCPWM_Initialize(void)

Files

Files

Name	Description
drv_mcpwm.h	MCPWM driver interface declarations for the static single instance driver.

Description

drv_mcpwm.h

MCPWM driver interface declarations for the static single instance driver.

Functions

	Name	Description
	DRV_MCPWM_Disable	Disables the MCPWM instance for the specified driver index. **Implementation:** Static
	DRV_MCPWM_Enable	Enables the MCPWM instance for the specified driver index. **Implementation:** Static
	DRV_MCPWM_Initialize	Initializes the MCPWM instance for the specified driver index. **Implementation:** Static

Description

Motor Control PWM (MCPWM) Driver Interface Declarations for Static Single Instance Driver

The MCPWM device driver provides a simple interface to manage the MCPWM module on Microchip microcontrollers. This file defines the interface Declarations for the MCPWM driver.

Remarks

Static interfaces incorporate the driver instance number within the names of the routines, eliminating the need for an object ID or object handle.

Static single-open interfaces also eliminate the need for the open handle.

File Name

drv_mcpwm.h

Company

Microchip Technology Inc.

NVM Driver Library

This section describes the Non-volatile Memory (NVM) Driver Library.

Migrating Applications from v1.03.01 and Earlier Releases of MPLAB Harmony

Provides information on migrating applications from v1.03.01 and earlier releases of MPLAB Harmony to release v1.04 and later.

Description

The NVM Driver Library APIs have changed beginning with the v1.04 release of MPLAB Harmony. Applications that were developed using the earlier version of the MPLAB Harmony NVM Driver (v1.03.01 and earlier) will not build unless the application calls to NVM Driver are updated. While the MHC utility provides an option to continue creating applications using the v1.03.01 and earlier NVM Driver API, it is recommended that existing applications migrate to the latest API to take advantage of the latest features in the NVM Driver. The following sections describe the API changes and other considerations while updating the application for changes in the NVM Driver.

All NVM Driver Demonstration Applications and NVM Driver related documentation have been updated to the latest (new) API. The following sections do not discuss changes in the NVM Driver configuration related code. This code is updated automatically when the project is regenerated using the MHC utility. Only the application related API changes are discussed.

The following table shows the beta API and corresponding v1.04 and Later MPLAB Harmony NVM Driver API.

v1.03.01 and Earlier NVM Driver API	v1.04 and Later NVM Driver API	v1.04 and Later API Notes
DRV_NVM_Initialize	DRV_NVM_Initialize	The init structure now has additional members that allow the NVM media address and geometry to be specified.
DRV_NVM_Deinitialize	DRV_NVM_Deinitialize	No change.
DRV_NVM_Status	DRV_NVM_Status	No change.
DRV_NVM_Open	DRV_NVM_Open	No change.
DRV_NVM_Close	DRV_NVM_Close	No change.
DRV_NVM_Read	DRV_NVM_Read	Parameters have changed: • Returns the command handle associated with the read operation as an output parameter • Data is now read in terms of blocks. The read block size is specified in the NVM Geometry.

DRV_NVM_Write	DRV_NVM_Write	Parameters have changed: • Returns the command handle associated with the write operation as an output parameter • Data is now written in terms of blocks. The write block size is specified in the NVM Geometry.
DRV_NVM_Erase	DRV_NVM_Erase	Parameters have changed: • Returns the command handle associated with the erase operation as an output parameter • NVM Flash is erased in terms of blocks. The erase block size is specified in the NVM Geometry.
DRV_NVM_EraseWrite	DRV_NVM_EraseWrite	Parameters have changed: • Returns the command handle associated with the Erase/Write operation as an output parameter. • Data is now written in terms of blocks. The write block size is specified in the NVM Geometry.
DRV_NVM_BlockEventHandlerSet	DRV_NVM_EventHandlerSet	Function name and parameter type have changed.
DRV_NVM_ClientStatus	Not Available	This API is no longer available.
DRV_NVM_BufferStatus	DRV_NVM_CommandStatus	The DRV_NVM_Read, DRV_NVM_Write, DRV_NVM_Erase, and DRV_NVM_EraseWrite functions now return a command handle associated with the operation. The status of the operation can be checked by passing the command handle to this function.
Not Available	DRV_NVM_GeometryGet	This API gives the following geometrical details of the NVM Flash: • Media Property • Number of Read/Write/Erase regions in the flash device • Number of Blocks and their size in each region of the device
Not Available	DRV_NVM_IsAttached	Returns the physical attach status of the NVM Flash.
Not Available	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM Flash.
Not Available	DRV_NVM_AddressGet	Returns the NVM Media Start address.

NVM Driver Initialization

DRV_NVM_INIT now takes the following two additional initialization parameters:

- mediaStartAddress - NVM Media Start address. The driver treats this address as the start address for read, write and erase operations.
- nvmMediaGeometry - Indicates the layout of the media in terms of read, write and erase regions.

The following code examples show how the driver initialization was performed with 1.03 APIs and how it is performed with the 1.04 APIs:

Example 1: v1.03 and Earlier Code

```
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
};

void SYS_Initialize (void *data)
{
.
.
    // Initialize NVM Driver Layer
    sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
.
}
```

Example: v1.04 and Later Code

```
/* NVM Geometry structure */
SYS_FS_MEDIA_REGION_GEOMETRY NVMGeometryTable[3] =
{
    {
        .blockSize = 1,
        .numBlocks = (DRV_NVM_MEDIA_SIZE * 1024),
    },
    {
       .blockSize = DRV_NVM_ROW_SIZE,
```

```
            .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_ROW_SIZE)
        },
        {
            .blockSize = DRV_NVM_PAGE_SIZE,
            .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_PAGE_SIZE)
        }
    };

    const SYS_FS_MEDIA_GEOMETRY NVMGeometry =
    {
        .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,
        .numReadRegions = 1,
        .numWriteRegions = 1,
        .numEraseRegions = 1,
        .geometryTable = (SYS_FS_MEDIA_REGION_GEOMETRY *)&NVMGeometryTable
    };

    const DRV_NVM_INIT drvNvmInit =
    {
        .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
        .nvmID = NVM_ID_0,
        .interruptSource = INT_SOURCE_FLASH_CONTROL,
        .mediaStartAddress = 0x9D010000,
        .nvmMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&NVMGeometry
    };

    void SYS_Initialize (void *data)
    {
    .
    .
        // Initialize NVM Driver Layer
        sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
    .
    .
    }
```

Addressing in NVM Driver

The v1.03.01 and earlier Read, Write, Erase and EraseWrite APIs took the actual address on which the operation was to be performed. The unit of access was bytes.

In v1.04 the addressing mechanism has been modified. The media start address is set in the DRV_NVM_Initialize. This address is used as the base address for the Read, Write, Erase and EraseWrite APIs. The unit of access is in terms of blocks. The NVM Geometry specifies the media layout in terms of:

- Number of erase, read and write regions
- Block size for erase, read and write operations.
- Number of blocks in erase, read and write regions

For example, in PIC32MZ family devices:

- Read block size = 1 byte
- Write block size = ROW Size = 2048 bytes
- Erase block size = PAGE Size = 16384 bytes

If the size of media is 32 KB then the following table illustrates the address range and number of blocks for the read, write and erase regions:

Region Type	Block Size	Number of blocks	Address range
Read Region	1 Byte	32 KB / Read block size = 32768	0–32767
Write Region	2048 Bytes	32 KB / Write block size = 16 blocks	0–15
Erase Region	16384 Bytes	32 KB / Erase block size = 2 blocks	0–1

Erasing Data on NVM Flash

The NVM Geometry indicates the number of erase blocks and the size of a single erase block. The Erase API takes in the erase block start address and the number of blocks to be erased. The following code examples show how to perform the erase operation in v1.03.01 and earlier and how to perform it with v1.04 and later.

Example: v1.03.01 and Earlier Code

```
DRV_HANDLE      myNVMHandle;    // Returned from DRV_NVM_Open
DRV_NVM_BUFFER_HANDLE bufferHandle;
```

```
bufferHandle = DRV_NVM_Erase(myNVMHandle, (uint8_t*)NVM_BASE_ADDRESS, DRV_NVM_PAGE_SIZE);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to erase NVM Media data */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Erase(nvmHandle, &nvmCommandHandle, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the erase request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Erase completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Erase Failed */
}
```

Writing Data to NVM Flash

The NVM Geometry indicates the number of write blocks and the size of a single write block. The Write API takes in the write block start address and the number of blocks to be written. The following code examples show how the write operation was performed in v1.03.01 and earlier and how to perform it with v1.04 and later APIs:

Example : v1.03.01 and Earlier Code

```
DRV_HANDLE myNVMHandle;    // Returned from DRV_NVM_Open
char       myBuffer[2 * DRV_NVM_ROW_SIZE];

// Destination address should be row aligned.
char       *destAddress = (char *)NVM_BASE_ADDRESS_TO_WRITE;

unsigned int    count = 2 * MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle  = DRV_NVM_Write(myNVMHandle, destAddress, &myBuffer[total], count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to write data to NVM Media */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint8_t writeBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Write(nvmHandle, &nvmCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Reading Data from NVM Flash

The NVM Geometry indicates the number of read blocks and the size of a single read block. The Read API takes in the read block start address and the number of blocks to be read. The following code examples show how the read operation was performed with v1.03.01 and earlier APIs and how to perform the same with v1.04 and later APIs:

Example: v1.03.01 and Earlier Code

```
DRV_HANDLE      myNVMHandle;    // Returned from DRV_NVM_Open
char            myBuffer[MY_BUFFER_SIZE];
char            *srcAddress = NVM_BASE_ADDRESS_TO_READ_FROM;
unsigned int    count = MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle  = DRV_NVM_Read(myNVMHandle, &myBuffer[total], srcAddress, count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to read data from NVM Media */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint8_t readBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = DRV_NVM_ROW_SIZE;

DRV_NVM_Read(nvmHandle, &nvmCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
```

```
{
    /* Failed to queue the read request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Introduction

The NVM Driver library provides APIs that can be used to interface with the NVM module (controller plus memory) for memory needs.

Description

The NVM Driver provides APIs for block access of the physical media through NVM Driver APIs. As shown in the NVM Driver Abstraction Model, an application or a client can access the physical media using multiple methods, which eventually are facilitated through the NVM Driver.

Memory Devices for PIC Microcontrollers

Depending on the device, there are two primary forms of on-chip memory: Programmable Flash memory and data EEPROM memory. The access mechanism for both of these types are varied.

Flash Program Memory

The Flash program memory is readable, writeable, and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete, restricting memory access, and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for program memory modification:

- Run-Time Self-Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

This section describes the RTSP techniques.

Using the Library

This topic describes the basic architecture of the NVM Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_nvm.h`

The interface to the NVM Driver Library is defined in the `drv_nvm.h` header file. Any C language source (`.c`) file that uses the NVM Driver library should include `drv_nvm.h`.

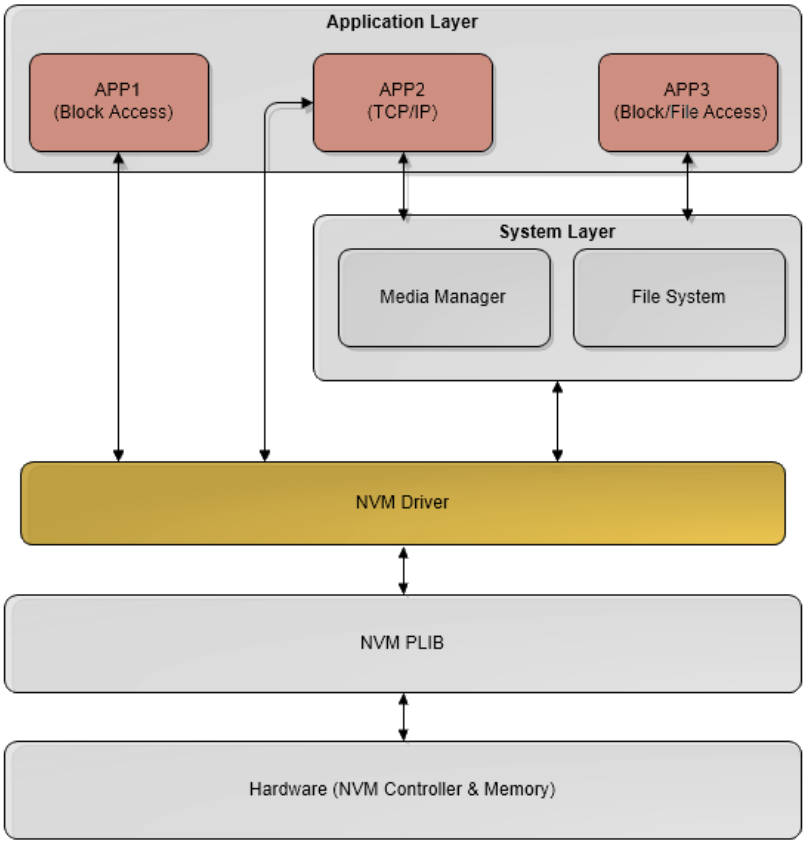
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

NVM Driver Abstraction Model



Abstraction Model

As shown in the previous diagram, the NVM Driver sits between the Peripheral Libraries and the application or system layer to facilitate block and file access to the NVM media (currently Flash). The application scenarios show how different layers can be accessed by different applications with certain needs. For example, APP1 can access the NVM Driver directly to erase, write, or read NVM with direct addressing. APP2, in this case TCP/IP, can bypass the system layer and access the NVM Driver layer if necessary to fulfill its robust data needs. Finally, APP3 accesses the NVM Driver through the File System Layer using block access methods, so the application does not need to keep track of the physical layout of the media.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the NVM module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Core Functions	Provides open, close, status and other setup functions.
Client Block Transfer Functions	Provides buffered data operation functions available in the core configuration.
Miscellaneous Functions	Provides driver miscellaneous functions related to versions and others.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality
- Media Functionality

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

NVM System Initialization

This section provides information for system initialization and reinitialization.

Description

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the NVM module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_NVM_INIT or by using initialization overrides) that are supported by the specific NVM device hardware:

- Device requested power state: One of the system module power states. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., NVM_ID_0)
- Defining the respective interrupt sources
- NVM Media Start Address
- NVM Media Geometry

The DRV_NVM_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_NVM_Deinitialize, DRV_NVM_Status, and DRV_NVM_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

The SYS_MODULE_INDEX is passed to the DRV_NVM_Initialize function to determine which type of memory is selected using: DRV_NVM_INDEX_0 - FLASH

Example:

```
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
    .mediaStartAddress = 0x9D010000,
    .nvmMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&NVMGeometry
};
void SYS_Initialize (void *data)
{
.
.
.

   // Initialize NVM Driver Layer
     sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
.
.
.
}
```

Tasks Routine

The system will call DRV_NVM_Tasks, from system task service (in a polled environment) or DRV_NVM_Tasks will be called from the Interrupt Service Routine (ISR) of the NVM.

Client Access Operation

This section provides information for general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_NVM_Open function. This provides the configuration required to open the NVM instance for operation. If the driver is deinitialized using the function DRV_NVM_Deinitialize, the application must call the DRV_NVM_Open function again to set up the instance of the NVM.

For the various options available for I/O INTENT please refer to **Data Types and Constants** in the Library Interface section.

Example:

```
    DRV_HANDLE handle;
```

```
handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Client Block Data Operation

This topic provides information on client block data operation.

Description

The NVM Driver provides a block interface to access the NVM media. The interface provides functionality to read, write, erase, and erase-write the NVM media. These interface functions depend on the block sizes and boundaries of the individual devices. The interfaces are responsible for keeping this information transparent from the application.

Erasing Data on the NVM:

The following steps outline the sequence for erasing data on the NVM media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Provide the block start address and the number of blocks to be erased and begin the erase process using the DRV_NVM_Erase.
4. The client can check the state of the erase request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the erase request.
5. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to erase NVM Media data
DRV_HANDLE             nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Erase(nvmHandle, &nvmCommandHandle, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the erase request. Handle the error. */
}

// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Erase completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Erase Failed */
}
```

Writing Data to the NVM:

The following steps outline the sequence to be followed for writing data to the NVM Media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. The client should ensure that blocks of addresses to which write is being performed should be in the erased state.
4. Provide the data to be written, block start address and the number of blocks to be written and begin write using the DRV_NVM_Write.

5. The client can check the state of the write request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the write request.
6. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to write data to NVM Media
DRV_HANDLE              nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint8_t writeBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Write(nvmHandle, &nvmCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}

// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Reading Data from the NVM:

The following steps outline the sequence to be followed for reading data from the NVM Media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Provide the target buffer, block start address and the number of blocks to be read and begin reading using the DRV_NVM_Read.
4. The client can check the state of the read request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the read request.
5. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to read data from NVM Media
DRV_HANDLE              nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint8_t readBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = DRV_NVM_ROW_SIZE;

DRV_NVM_Read(nvmHandle, &nvmCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the read request. Handle the error. */
}
```

```
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Configuring the Library

Macros

	Name	Description
	DRV_NVM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_NVM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_NVM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_NVM_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	DRV_NVM_ROW_SIZE	Specifies the NVM Driver Program Row Size in bytes.
	DRV_NVM_ERASE_WRITE_ENABLE	Enables support for NVM Driver Erase Write Feature.
	DRV_NVM_PAGE_SIZE	Specifies the NVM Driver Program Page Size in bytes.
	DRV_NVM_DISABLE_ERROR_CHECK	Disables the error checks in the driver.
	DRV_NVM_MEDIA_SIZE	Specifies the NVM Media size.
	DRV_NVM_MEDIA_START_ADDRESS	Specifies the NVM Media start address.
	DRV_NVM_SYS_FS_REGISTER	Register to use with the File system

Description

The configuration of the NVM Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the NVM Driver. Based on the selections made, the NVM Driver may support the selected features. These configuration settings will apply to all instances of the NVM Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_NVM_BUFFER_OBJECT_NUMBER Macro

Selects the maximum number of buffer objects

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_BUFFER_OBJECT_NUMBER 5
```

Description

NVM Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The NVM Driver can queue up DRV_NVM_BUFFER_OBJECT_NUMBER of read/write/erase requests before return a DRV_NVM_BUFFER_HANDLE_INVALID due to the queue being full. Buffer objects are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_CLIENTS_NUMBER Macro

Selects the maximum number of clients

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_CLIENTS_NUMBER 1
```

Description

NVM maximum number of clients

This definition selects the maximum number of clients that the NVM driver can supported at run time. This constant defines the total number of NVM driver clients that will be available to all instances of the NVM driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_INSTANCES_NUMBER Macro

Selects the maximum number of Driver instances that can be supported by the dynamic driver.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_INSTANCES_NUMBER 1
```

Description

NVM Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_INTERRUPT_MODE Macro

Macro specifies operation of the driver to be in the interrupt mode or polled mode

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_INTERRUPT_MODE true
```

Description

NVM interrupt and polled mode operation control

This macro specifies operation of the driver to be in the interrupt mode or polled mode

- true - Select if interrupt mode of NVM operation is desired
- false - Select if polling mode of NVM operation is desired

Not defining this option to true or false will result in build error.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_ROW_SIZE Macro

Specifies the NVM Driver Program Row Size in bytes.

File

drv_nvm.h

C

```
#define DRV_NVM_ROW_SIZE (NVM_ROW_SIZE)
```

Description

NVM Driver Program Row Size.

This definition specifies the NVM Driver Program Row Size in bytes. This parameter is device specific and is obtained from the device specific processor header file. The Program Row Size is the minimum block size that can be programmed in one program operation.

Remarks

None

DRV_NVM_ERASE_WRITE_ENABLE Macro

Enables support for NVM Driver Erase Write Feature.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_ERASE_WRITE_ENABLE
```

Description

NVM Driver Erase Write Feature Enable

Specifying this macro enable row erase write feature. If this macro is specified, the drv_nvm_erasewrite.c file should be added in the project. Support for DRV_NVM_EraseWrite() function then gets enabled.

Remarks

This macro is optional and should be specified only if the DRV_NVM_EraseWrite() function is required.

DRV_NVM_PAGE_SIZE Macro

Specifies the NVM Driver Program Page Size in bytes.

File

drv_nvm.h

C

```
#define DRV_NVM_PAGE_SIZE (NVM_PAGE_SIZE)
```

Description

NVM Driver Program Page Size.

This definition specifies the NVM Driver Program Page Size in bytes. This parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

DRV_NVM_DISABLE_ERROR_CHECK Macro

Disables the error checks in the driver.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_DISABLE_ERROR_CHECK
```

Description

NVM Driver Disable Error Checks

Specifying this macro disables the error checks in the driver. Error checks like parameter validation, NULL checks etc, will be disabled in the driver in order to optimize the code space.

Remarks

This macro is optional and should be specified only if code space is a constraint.

DRV_NVM_MEDIA_SIZE Macro

Specifies the NVM Media size.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_MEDIA_SIZE 32
```

Description

NVM Media Size

This definition specifies the NVM Media Size to be used. The size is specified in number of Kilo Bytes. The media size MUST never exceed physical available NVM Memory size. Application code requirements should be kept in mind while defining this parameter.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_MEDIA_START_ADDRESS Macro

Specifies the NVM Media start address.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_MEDIA_START_ADDRESS 0x9D010000
```

Description

NVM Media Start Address

This definition specifies the NVM Media Start address parameter.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_SYS_FS_REGISTER
```

Description

NVM Driver Register with File System

Specifying this macro enables the NVM driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the NVM driver is to be used with the File System.

Building the Library

This section lists the files that are available in the NVM Driver Library.

Description

This section list the files that are available in the `\src` folder of the NVM Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/nvm`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_nvm.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_nvm.c` `/src/dynamic/drv_nvm_erasewrite.c`	Dynamic NVM Driver implementation file. Dynamic NVM Driver Erase/Write implementation file.
`/src/static/drv_nvm_static.c`	Static NVM Driver implementation file for single clients.
`/src/static_multi/drv_nvm_static_multi.c`	Static NVM Driver implementation file for multiple clients.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The NVM Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_NVM_Initialize	Initializes the NVM instance for the specified driver index **Implementation:** Static/Dynamic
	DRV_NVM_Deinitialize	Deinitializes the specified instance of the NVM driver module **Implementation:** Static/Dynamic
	DRV_NVM_Status	Gets the current status of the NVM driver module. **Implementation:** Static/Dynamic

b) Client Core Functions

	Name	Description
	DRV_NVM_Open	Opens the specified NVM driver instance and returns a handle to it **Implementation:** Static/Dynamic
	DRV_NVM_Close	Closes an opened-instance of the NVM driver **Implementation:** Static/Dynamic

	DRV_NVM_Read	Reads blocks of data from the specified address in memory. **Implementation:** Static/Dynamic
	DRV_NVM_Write	Writes blocks of data starting from the specified address in flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_Erase	Erase the specified number of blocks of the Flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EraseWrite	Erase and Write blocks of data starting from a specified address in flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Static/Dynamic

c) Client Block Data Functions

	Name	Description
	DRV_NVM_Tasks	Maintains the driver's erase and write state machine and implements its ISR. **Implementation:** Static/Dynamic

d) Status Functions

	Name	Description
	DRV_NVM_AddressGet	Returns the NVM media start address **Implementation:** Static/Dynamic
	DRV_NVM_CommandStatus	Gets the current status of the command. **Implementation:** Static/Dynamic
	DRV_NVM_GeometryGet	Returns the geometry of the device. **Implementation:** Static/Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_NVM_IsAttached	Returns the physical attach status of the NVM. **Implementation:** Static/Dynamic
	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM. **Implementation:** Static/Dynamic

f) Data Types and Constants

	Name	Description
	DRV_NVM_INDEX_0	NVM driver index definitions
	DRV_NVM_INIT	Defines the data required to initialize or reinitialize the NVM driver
	DRV_NVM_INDEX_1	This is macro DRV_NVM_INDEX_1.
	DRV_NVM_EVENT	Identifies the possible events that can result from a request.
	DRV_NVM_EVENT_HANDLER	Pointer to a NVM Driver Event handler function
	DRV_NVM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_NVM_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_NVM_COMMAND_HANDLE_INVALID	This value defines the NVM Driver's Invalid Command Handle.

Description

This section describes the Application Programming Interface (API) functions of the NVM Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_NVM_Initialize Function

Initializes the NVM instance for the specified driver index

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_MODULE_OBJ DRV_NVM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the NVM driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other NVM routine is called.

This routine should only be called once during system initialization unless DRV_NVM_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_NVM_Status operation. The system must use DRV_NVM_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Preconditions

None.

Example

```
// This code snippet shows an example
// of initializing the NVM Driver.

SYS_MODULE_OBJ  objectHandle;

SYS_FS_MEDIA_REGION_GEOMETRY gNvmGeometryTable[3] =
{
    {
        // Read Region Geometry
        .blockSize = 1,
        .numBlocks = (DRV_NVM_MEDIA_SIZE * 1024),
    },
    {
        // Write Region Geometry
        .blockSize = DRV_NVM_ROW_SIZE,
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_ROW_SIZE)
    },
    {
        // Erase Region Geometry
        .blockSize = DRV_NVM_PAGE_SIZE,
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_PAGE_SIZE)
    }
};

const SYS_FS_MEDIA_GEOMETRY gNvmGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,

    // Number of read, write and erase entries in the table
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = &gNvmGeometryTable
};

// FLASH Driver Initialization Data
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
    .mediaStartAddress = NVM_BASE_ADDRESS,
    .nvmMediaGeometry = &gNvmGeometry
};
```

```
//usage of DRV_NVM_INDEX_0 indicates usage of Flash-related APIs
objectHandle = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT*)&drvNVMInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized also the type of memory used
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_NVM_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_NVM_Deinitialize Function

Deinitializes the specified instance of the NVM driver module

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the NVM driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_NVM_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_NVM_Initialize routine

Example

```
// This code snippet shows an example
// of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_NVM_Initialize
SYS_STATUS          status;

DRV_NVM_Deinitialize(object);

status = DRV_NVM_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_NVM_Deinitialize

(

SYS_MODULE_OBJ object
);

DRV_NVM_Status Function

Gets the current status of the NVM driver module.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_STATUS DRV_NVM_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations.
SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the NVM driver module.

Remarks

This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_NVM_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_NVM_Initialize
SYS_STATUS          NVMStatus;

NVMStatus = DRV_NVM_Status(object);
else if (SYS_STATUS_ERROR >= NVMStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_NVM_Initialize routine

Function

SYS_STATUS DRV_NVM_Status
(
SYS_MODULE_OBJ object
);

b) Client Core Functions

DRV_NVM_Open Function

Opens the specified NVM driver instance and returns a handle to it

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
DRV_HANDLE DRV_NVM_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_NVM_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified NVM driver instance and provides a handle. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_NVM_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has has already been opened, it cannot be opened exclusively.

Preconditions

Function DRV_NVM_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_NVM_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

DRV_NVM_Close Function

Closes an opened-instance of the NVM driver

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the NVM driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_NVM_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close

operation has completed.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

DRV_NVM_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_NVM_Open

DRV_NVM_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_NVM_Close

(

const DRV_HANDLE handle

);

DRV_NVM_Read Function

Reads blocks of data from the specified address in memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Read(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine reads blocks of data from the specified address in memory. This operation is blocking and returns with the required data in the target buffer. If an event handler is registered with the driver the event handler would be invoked from within this function to indicate the status of the operation. This function should not be used to read areas of memory which are queued to be programmed or erased. If required, the program or erase operations should be allowed to complete. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the target buffer pointer is NULL
- if the number of blocks to be read is zero or more than the actual number of blocks available
- if a buffer object could not be allocated to the request
- if the client opened the driver in write only mode

Remarks

None.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

DRV_NVM_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_READ_FROM;
```

```
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

DRV_NVM_Read(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read Successful
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
targetBuffer	Buffer into which the data read from the NVM Flash instance will be placed
blockStart	Start block address in NVM memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_NVM_Read

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

void * targetBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_NVM_Write Function

Writes blocks of data starting from the specified address in flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Write(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void * sourceBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the source buffer pointer is NULL
- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available

- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_NVM_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

Performing a flash programming operation while executing (fetching) instructions from program Flash memory, the CPU stalls (waits) until the programming operation is finished. The CPU will not execute any instruction, or respond to interrupts, during this time. If any interrupts occur during the programming cycle, they remain pending until the cycle completes. This makes the NVM write operation blocking in nature.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

DRV_NVM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

The flash address location which has to be written, must have be erased before using the DRV_NVM_Erase() routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Write(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into NVM Flash
blockStart	Start block address of NVM Flash where the write should begin. This address should be aligned on a block boundary.
nBlock	Total number of blocks to be written.

Function

void DRV_NVM_Write
(
const DRV_HANDLE handle,
DRV_NVM_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_NVM_Erase Function

Erase the specified number of blocks of the Flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Erase(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, uint32_t blockStart,
uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation of flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the client opened the driver for read only
- if the number of blocks to be erased is either zero or more than the number of blocks actually available
- if the erase queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the erase operation was successful or DRV_NVM_EVENT_COMMAND_ERROR event if the erase operation was not successful.

Remarks

Performing a flash erase operation while executing (fetching) instructions from program Flash memory, the CPU stalls (waits) until the erase operation is finished. The CPU will not execute any instruction, or respond to interrupts, during this time. If any interrupts occur during the programming cycle, they remain pending until the cycle completes. This make the NVM erase operation blocking in nature.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE to obtain a valid opened device handle.

Example

```
// Destination address should be block aligned.
```

```
uint32_t blockStart;
uint32_t nBlock;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Erase( myNVMHandle, &commandHandle, blockStart, nBlock );

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in NVM memory from where the erase should begin. This should be aligned on a DRV_NVM_PAGE_SIZE byte boundary.
nBlock	Total number of blocks to be erased.

Function

void DRV_NVM_Erase

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,

uint32_t nBlock

);

DRV_NVM_EraseWrite Function

Erase and Write blocks of data starting from a specified address in flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_EraseWrite(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void *
sourceBuffer, uint32_t writeBlockStart, uint32_t nWriteBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function combines the step of erasing a page and then writing the row. The application can use this function if it wants to avoid having to explicitly delete a page in order to update the rows contained in the page.

This function schedules a non-blocking operation to erase and write blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_NVM_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In order to use this function, the DRV_NVM_ERASE_WRITE_ENABLE must be defined in system_config.h and the drv_nvm_erasewrite.c file must be included in the project.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE as a parameter to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_EraseWrite(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
```

```
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle. If NULL, then buffer handle is not returned.
sourceBuffer	The source buffer containing data to be programmed into NVM Flash
writeBlockStart	Start block address of NVM Flash where the write should begin. This address should be aligned on a DRV_NVM_ROW_SIZE byte boundary.
nWriteBlock	Total number of blocks to be written.

Function

void DRV_NVM_EraseWrite

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

void * sourceBuffer,

uint32_t writeBlockStart,

uint32_t nWriteBlock

);

DRV_NVM_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls a write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any write or erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_NVM_COMMAND_HANDLE commandHandle;

// drvNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver. This is done once.

DRV_NVM_EventHandlerSet(drvNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Read(drvNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_NVM_EventHandlerSet
(
const DRV_HANDLE handle,
const void * eventHandler,
const uintptr_t context
);

c) Client Block Data Functions

DRV_NVM_Tasks Function

Maintains the driver's erase and write state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal write and erase state machine and implement its ISR for interrupt-driven implementations.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_NVM_Initialize

while (true)
{
    DRV_NVM_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_NVM_Initialize)

Function

void DRV_NVM_Tasks
(
SYS_MODULE_OBJ object
);

d) Status Functions

DRV_NVM_AddressGet Function

Returns the NVM media start address

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
uintptr_t DRV_NVM_AddressGet(const DRV_HANDLE handle);
```

Returns

Start address of the NVM Media if the handle is valid otherwise NULL.

Description

This function returns the NVM Media start address.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
uintptr_t startAddress;
startAddress = DRV_NVM_AddressGet(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

uintptr_t DRV_NVM_AddressGet

(

const DRV_HANDLE handle

);

DRV_NVM_CommandStatus Function

Gets the current status of the command.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
DRV_NVM_COMMAND_STATUS DRV_NVM_CommandStatus(const DRV_HANDLE handle, const DRV_NVM_COMMAND_HANDLE
commandHandle);
```

Returns

A DRV_NVM_COMMAND_STATUS value describing the current status of the command. Returns DRV_NVM_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to polled on. The function may return DRV_NVM_COMMAND_HANDLE_INVALID in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another erase or write request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive write or erase operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_NVM_Initialize() routine must have been called.

The DRV_NVM_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                  handle;         // Returned from DRV_NVM_Open
DRV_NVM_COMMAND_HANDLE      commandHandle;
DRV_NVM_COMMAND_STATUS      status;

status = DRV_NVM_CommandStatus(handle, commandHandle);
if(status == DRV_NVM_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_NVM_COMMAND_STATUS DRV_NVM_CommandStatus
(
const DRV_HANDLE handle,
const DRV_NVM_COMMAND_HANDLE commandHandle
);

DRV_NVM_GeometryGet Function

Returns the geometry of the device.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_NVM_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the NVM Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * nvmFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;
```

```
nvmFlashGeometry = DRV_NVM_GeometryGet(nvmOpenHandle1);

readBlockSize  = nvmFlashGeometry->geometryTable->blockSize;
nReadBlocks = nvmFlashGeometry->geometryTable->numBlocks;
nReadRegions = nvmFlashGeometry->numReadRegions;

writeBlockSize  = (nvmFlashGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (nvmFlashGeometry->geometryTable +2)->blockSize;

totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_NVM_GeometryGet

(

const DRV_HANDLE handle

);

e) Miscellaneous Functions

DRV_NVM_IsAttached Function

Returns the physical attach status of the NVM.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
bool DRV_NVM_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns true.

Description

This function returns the physical attach status of the NVM.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The NVM media is always attached and so the below
// always returns true.

bool isNVMAttached;
isNVMAttached = DRV_NVM_isAttached(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_NVM_IsAttached

(

const DRV_HANDLE handle
);

DRV_NVM_IsWriteProtected Function

Returns the write protect status of the NVM.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
bool DRV_NVM_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Always returns false.

Description

This function returns the physical attach status of the NVM. This function always returns false.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The NVM media is treated as always writeable.
bool isWriteProtected;
isWriteProtected = DRV_NVM_IsWriteProtected(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_NVM_IsWriteProtected
(
const DRV_HANDLE handle
);

f) Data Types and Constants

DRV_NVM_INDEX_0 Macro

NVM driver index definitions

File

drv_nvm.h

C

```
#define DRV_NVM_INDEX_0 0
```

Description

Driver NVM Module Index reference

These constants provide NVM driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_NVM_Initialize and

DRV_NVM_Open routines to identify the driver instance in use.

DRV_NVM_INIT Structure

Defines the data required to initialize or reinitialize the NVM driver

File

drv_nvm.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  NVM_MODULE_ID nvmID;
  INT_SOURCE interruptSource;
  uint32_t mediaStartAddress;
  const SYS_FS_MEDIA_GEOMETRY * nvmMediaGeometry;
} DRV_NVM_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
NVM_MODULE_ID nvmID;	Identifies NVM hardware module (PLIB-level) ID
INT_SOURCE interruptSource;	Interrupt Source for Write Interrupt
uint32_t mediaStartAddress;	NVM Media start address. The driver treats this address as • block 0 address for read, write and erase operations.
const SYS_FS_MEDIA_GEOMETRY * nvmMediaGeometry;	NVM Media geometry object.

Description

NVM Driver Initialization Data

This data type defines the data required to initialize or reinitialize the NVM driver.

Remarks

Not all initialization features are available for all devices. Please refer to the specific device data sheet to determine availability.

DRV_NVM_INDEX_1 Macro

File

drv_nvm.h

C

```
#define DRV_NVM_INDEX_1 1
```

Description

This is macro DRV_NVM_INDEX_1.

DRV_NVM_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_nvm.h

C

```
typedef enum {
  DRV_NVM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_NVM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_NVM_EVENT;
```

Members

Members	Description
DRV_NVM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_NVM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

NVM Driver Events

This enumeration identifies the possible events that can result from a Write or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_NVM_EventHandlerSet function when a request is completed.

DRV_NVM_EVENT_HANDLER Type

Pointer to a NVM Driver Event handler function

File

drv_nvm.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_NVM_EVENT_HANDLER;
```

Returns

None.

Description

NVM Driver Event Handler Function Pointer

This data type defines the required function signature for the NVM event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_NVM_EVENT_COMMAND_COMPLETE, it means that the write or a erase operation was completed successfully.

If the event is DRV_NVM_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_NVM_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations within this function.

Example

```
void APP_MyNvmEventHandler
(
    DRV_NVM_EVENT event,
    DRV_NVM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;
```

```
            case DRV_NVM_EVENT_COMMAND_ERROR:
            default:

                // Handle error.
                break;
        }
    }
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_NVM_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_nvm.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_NVM_COMMAND_HANDLE;
```

Description

NVM Driver command handle.

A command handle is returned by a call to the Read, Write or Erase functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_NVM_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_nvm.h

C

```
typedef enum {
  DRV_NVM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_NVM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_NVM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_NVM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_NVM_COMMAND_STATUS;
```

Members

Members	Description
DRV_NVM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_NVM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_NVM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_NVM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

NVM Driver Command Status

NVM Driver command Status

This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_NVM_COMMAND_HANDLE_INVALID Macro

This value defines the NVM Driver's Invalid Command Handle.

File

drv_nvm.h

C

```
#define DRV_NVM_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

NVM Driver Invalid Command Handle.

This value defines the NVM Driver Invalid Command Handle. This value is returned by read/write/erase routines when the command request was not accepted.

Remarks

None.

Files

Files

Name	Description
drv_nvm.h	NVM Driver Interface Definition
drv_nvm_config_template.h	NVM driver configuration definitions.

Description

This section lists the source and header files used by the NVM Driver Library.

drv_nvm.h

NVM Driver Interface Definition

Enumerations

	Name	Description
	DRV_NVM_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_NVM_EVENT	Identifies the possible events that can result from a request.

Functions

	Name	Description
	DRV_NVM_AddressGet	Returns the NVM media start address **Implementation:** Static/Dynamic
	DRV_NVM_Close	Closes an opened-instance of the NVM driver **Implementation:** Static/Dynamic
	DRV_NVM_CommandStatus	Gets the current status of the command. **Implementation:** Static/Dynamic
	DRV_NVM_Deinitialize	Deinitializes the specified instance of the NVM driver module **Implementation:** Static/Dynamic
	DRV_NVM_Erase	Erase the specified number of blocks of the Flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EraseWrite	Erase and Write blocks of data starting from a specified address in flash memory. **Implementation:** Static/Dynamic

	DRV_NVM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Static/Dynamic
	DRV_NVM_GeometryGet	Returns the geometry of the device. **Implementation:** Static/Dynamic
	DRV_NVM_Initialize	Initializes the NVM instance for the specified driver index **Implementation:** Static/Dynamic
	DRV_NVM_IsAttached	Returns the physical attach status of the NVM. **Implementation:** Static/Dynamic
	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM. **Implementation:** Static/Dynamic
	DRV_NVM_Open	Opens the specified NVM driver instance and returns a handle to it **Implementation:** Static/Dynamic
	DRV_NVM_Read	Reads blocks of data from the specified address in memory. **Implementation:** Static/Dynamic
	DRV_NVM_Status	Gets the current status of the NVM driver module. **Implementation:** Static/Dynamic
	DRV_NVM_Tasks	Maintains the driver's erase and write state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_NVM_Write	Writes blocks of data starting from the specified address in flash memory. **Implementation:** Static/Dynamic

Macros

	Name	Description
	DRV_NVM_COMMAND_HANDLE_INVALID	This value defines the NVM Driver's Invalid Command Handle.
	DRV_NVM_INDEX_0	NVM driver index definitions
	DRV_NVM_INDEX_1	This is macro DRV_NVM_INDEX_1.
	DRV_NVM_PAGE_SIZE	Specifies the NVM Driver Program Page Size in bytes.
	DRV_NVM_ROW_SIZE	Specifies the NVM Driver Program Row Size in bytes.

Structures

	Name	Description
	DRV_NVM_INIT	Defines the data required to initialize or reinitialize the NVM driver

Types

	Name	Description
	DRV_NVM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_NVM_EVENT_HANDLER	Pointer to a NVM Driver Event handler function

Description

NVM Driver Interface Definition

The NVM driver provides a simple interface to manage the Non Volatile Flash Memory on Microchip microcontrollers. This file defines the interface definition for the NVM driver.

File Name

drv_nvm.h

Company

Microchip Technology Inc.

drv_nvm_config_template.h

NVM driver configuration definitions.

Macros

	Name	Description
	DRV_NVM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_NVM_CLIENTS_NUMBER	Selects the maximum number of clients

	DRV_NVM_DISABLE_ERROR_CHECK	Disables the error checks in the driver.
	DRV_NVM_ERASE_WRITE_ENABLE	Enables support for NVM Driver Erase Write Feature.
	DRV_NVM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_NVM_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	DRV_NVM_MEDIA_SIZE	Specifies the NVM Media size.
	DRV_NVM_MEDIA_START_ADDRESS	Specifies the NVM Media start address.
	DRV_NVM_SYS_FS_REGISTER	Register to use with the File system

Description

NVM Driver Configuration Template Header file.

This template file describes all the mandatory and optional configuration macros that are needed for building the NVM driver. Do not include this file in source code.

File Name

drv_nvm_config_template.h

Company

Microchip Technology Inc.

Output Compare Driver Library

This section describes the Output Compare Driver Library.

Introduction

The Output Compare Static Driver provides a high-level interface to manage the Output Compare module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Enabling/Disabling of the output compare
- Starting/Stopping of the output compare
- Fault checking

Library Interface

Functions

	Name	Description
	DRV_OC_Disable	Disables the Output Compare instance for the specified driver index. **Implementation:** Static
	DRV_OC_Enable	Enables the Output Compare for the specified driver index. **Implementation:** Static
	DRV_OC_FaultHasOccurred	Checks if a Fault has occurred for the specified driver index. **Implementation:** Static
	DRV_OC_Initialize	Initializes the Comparator instance for the specified driver index. **Implementation:** Static
	DRV_OC_Start	Starts the Comparator instance for the specified driver index. **Implementation:** Static
	DRV_OC_Stop	Stops the Output Compare instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Output Compare Driver Library.

Functions

DRV_OC_Disable Function

Disables the Output Compare instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Disable();
```

Returns

None.

Description

This routine disables the Output Compare for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Disable(void)

DRV_OC_Enable Function

Enables the Output Compare for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Enable();
```

Returns

None.

Description

This routine enables the Output Compare for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Enable(void)

DRV_OC_FaultHasOccurred Function

Checks if a Fault has occurred for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
bool DRV_OC_FaultHasOccurred();
```

Returns

Boolean

- 1 - A Fault has occurred
- 0 - A Fault has not occurred

Description

This routine checks whether or not a Fault has occurred for the specified driver index. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

bool DRV_OC_FaultHasOccurred(void)

DRV_OC_Initialize Function

Initializes the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Initialize();
```

Returns

None.

Description

This routine initializes the Output Compare driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Output Compare module ID. For example, driver instance 0 can be assigned to Output Compare 1.

Remarks

This routine must be called before any other Comparator routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_OC_Initialize(void)

DRV_OC_Start Function

Starts the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Start();
```

Returns

None.

Description

This routine starts the Output Compare for the specified driver instance.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Start(void)

DRV_OC_Stop Function

Stops the Output Compare instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Stop();
```

Returns

None.

Description

This routine stops the Output Compare for the specified driver instance.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Stop(void)

Parallel Master Port (PMP) Driver Library

This section describes the Parallel Master Port Driver Library.

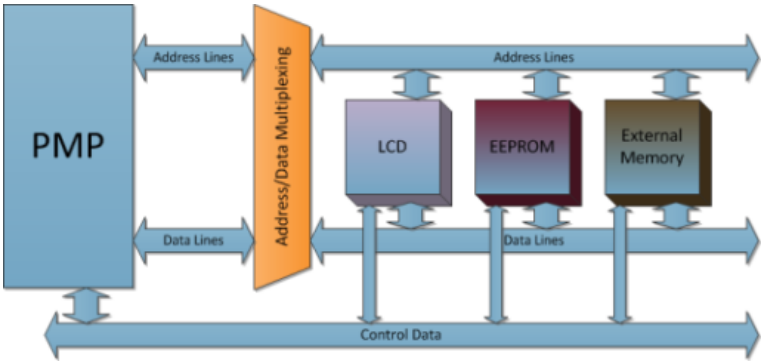
Introduction

This library provides an interface to manage the Parallel Master Port (PMP) module on Microchip family of microcontrollers in different modes of operation.

Description

The Parallel Master Port (PMP) is a parallel 8-bit/16-bit I/O module specifically designed to communicate with a wide variety of parallel devices such as communications peripherals, LCDs, external memory devices and microcontrollers. Because the interfaces to parallel peripherals vary significantly, the PMP module is highly configurable.

The following figure shows a generic block diagram, which illustrates the ways the PMP module can be used:



The PMP module can be used in different modes. Master and Slave are the two modes that can have additional sub-modes, depending on the different microcontroller families.

Master Mode: In Master mode, the PMP module can provide a 8-bit or 16-bit data bus, up to 16 bits of address, and all of the necessary control signals to operate a variety of external parallel devices such as memory devices, peripherals and slave microcontrollers. The PMP master modes provide a simple interface for reading and writing data, but not executing program instructions from external devices, such as SRAM or Flash memories.

Slave Mode: Slave mode only supports 8-bit data and the module control pins are automatically dedicated when this mode is selected.

Using the Library

This topic describes the basic architecture of the PMP Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_pmp.h`

The interface to the PMP Driver library is defined in the `drv_pmp.h` header file. This file is included by the `drv.h` file. Any C language source (`.c`) file that uses the PMP Driver Library should include `drv.h`.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

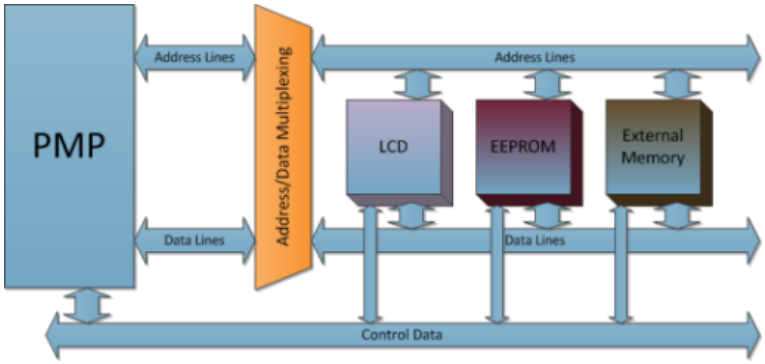
This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Description

Depending on the device,the PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.

The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.



PMP Hardware Abstraction Model Diagram

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the PMP module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks, and status functions.
Client Interaction Functions	Provides open, close, client status and client mode configuration functions.
Client Transfer Functions	Provides interface for data transfer in master and slave mode.
Miscellaneous	Provides driver miscellaneous functions, version identification functions, etc.

How the Library Works

This section describes how the PMP Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration). Refer to the Configuring the Library section for more details on how to configure the driver.

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the PMP Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

The following is the sequence in which various routines should be called:

1. Call DRV_PMP_Initialize to initialize the PMP Driver. Note that this may be performed by the MPLAB Harmony system module. The DRV_PMP_Status function may be used to check the status of the initialization.
2. Once initialization for a particular driver instance is done, the client wanting to use the driver can open it using DRV_PMP_Open.
3. The DRV_PMP_ModeConfig function should now be called, which will configure the driver for the exact mode of operation required by that client.
4. After configuring the mode, DRV_PMP_Write and/or DRV_PMP_Read can be called by the user application to Write/Read using the PMP module. Calling these functions does not start the PMP transfer immediately in non-interrupt mode. Instead, all of these transfer tasks are queued in an internal queue. Actual transfer starts only when the PMP Task function is called by the system/user. In interrupt mode, although transfer tasks are queued, the actual transfer starts immediately.
5. PMP Write and Read functions return an ID of that particular transfer, which should be saved by user to get the status of that transfer later.
6. The system will either call DRV_PMP_Tasks from the System Task Service (in a polled environment), or it will be called from the ISR of the PMP.
7. At any time status of the transfer can be obtained by using DRV_PMP_TransferStatus.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

System Initialization

This section describes initialization and reinitialization features.

Description

Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the PMP device will be initialized with the following configuration settings:

Initialization Member	Description
`moduleInit`	System module initialization of the power state.
`pmpId`	PMP hardware module ID (peripheral library-level ID).
`stopInIdle`	Decide whether or not the module should be stopped in Idle mode.
`muxMode`	To select one of the different multiplexing modes possible for PMP module.
`inputBuffer`	Select the type of Input Buffer (TTL or Schmitt Trigger).
`polarity`	Select polarity of different PMP pins.
`ports`	Set the pins the user wants to use as port or PMP pins.

The DRV_PMP_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_PMP_Reinitialize, DRV_PMP_Deinitialize, DRV_PMP_Status, and DRV_PMP_Tasks.

Example for PMP Initialization Through the DRV_PMP_INIT Structure

```
DRV_PMP_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       pmpStatus;

// populate the PMP init configuration structure
    init.inputBuffer = PMP_INPUT_BUFFER_TTL;
    init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
    init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
    init.ports.readWriteStrobe = PORT_ENABLE;
    init.ports.writeEnableStrobe = PORT_ENABLE;
    init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
    init.pmpID            = PMP_ID_0;
    init.stopInIdle       = false;
    init.muxMode          = PMP_MUX_NONE;

object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

pmpStatus = DRV_PMP_Status(object);

if ( SYS_STATUS_READY != pmpStatus)
{
    // Handle error
}
```

Deinitialization

Once the initialize operation has been called, the deinitialize operation must be called before the initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the function will never block for hardware PMP access. If the operation requires time to allow the hardware to complete, which will be reported by DRV_PMP_Status.

Status

PMP status is available to query the module state before, during and after initialization, deinitialization, and reinitialization.

Tasks Routine

The DRV_PMP_Tasks function will see the queue status and perform the task of transferring the data accordingly. In the Blocking mode when interrupts are disabled, it will finish one of the tasks completely (that means emptying one space in queue), and then return back. Whereas in Non-Blocking mode, it will return back just after starting one word (8-bit or 16-bit) of transfer (may not be emptying one space in the queue, as that task may not be completely finished).

The DRV_PMP_Tasks function can be called in two ways:

- By the system task service in a polled environment
- By the ISR of the PMP in an interrupt-based system

Example: Polling

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1)
    {
        DRV_PMP_Tasks (object);
    }
}
```

Example: Interrupt

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );
```

```
    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1);
}

/* Sample interrupt routine not specific to any device family */
void ISR PMPInterrupt(void)
{
    //Call the PMP Tasks routine
    DRV_PMP_Tasks(object);
}
```

Note: A PMP transfer in Blocking mode in an interrupt environment is not supported.

Transfer Operation

This section describes transfer operation.

Description

Once the PMP Driver is open and configured for a client, it is set to start Reading/Writing through DRV_PMP_Read and DRV_PMP_Write. However, these functions will not directly start reading or writing. These will just put the relevant information in a queue in non-interrupt mode and return an ID that can be used later for checking the transfer status. In Interrupt mode, the Read/Write functions will trigger the transfer immediately after storing the transfer information in the queue.

The user must use a buffer pointing to character for data values.

The repeatCount parameter allows the user to repeatedly write the same nBytes of data into the slave devices.

Example:

```
unsigned char myReadBuffer[300], myWriteBuffer[100]; // has to be 'char' arrays
uint32_t deviceAddress, nBytes, repeatCount, i;
uint32_t writeID, readID;
DRV_HANDLE handle;

//initialize, open and configure the driver/client
/* ... */

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
   myWriteBuffer[i]=i*5+7;
}

/* it will write 100 bytes of data in the location starting from 0x0206 and then it will repeat
 writing the same set of data in next 100 location starting from 0x206+100 for 8 bit data mode
 and 50 location starting from 0x206+50 for 16 bit data mode. */
writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);

// it will read 300 locations starting from 0x0206 into myReadBuffer
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes);
```

Transfer Status

The status of the read/write transfers can be obtained using API DRV_PMP_TransferStatus.

Example:

```
DRV_PMP_TRANSFER_STATUS writeStatus, readStatus;
uint32_t writeID, readID;

writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);
```

Client Operation

This section describes general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_PMP_Open function with a specific intent. This provides the configuration required to open the PMP instance for operation. If the driver is deinitialized using the function DRV_PMP_Deinitialize, the application must call the DRV_PMP_Open function again to set up the instance of the PMP. The function DRV_PMP_Open need not be called again if the system is reinitialized using the DRV_PMP_Reinitialize function.

The PMP driver supports DRV_IO_INTENT_NONBLOCKING, DRV_IO_INTENT_BLOCKING, DRV_IO_INTENT_EXCLUSIVE, and DRV_IO_INTENT_SHARED IO.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}
```

The function DRV_PMP_Close closes an already opened instance of the PMP driver, invalidating the handle. DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

/*...*/

DRV_PMP_Close( handle );
```

The client has the option to check the status through the function DRV_PMP_ClientStatus.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if ( DRV_PMP_CLIENT_STATUS_OPEN != DRV_PMP_ClientStatus( handle ) )
   return 0;
```

Client Mode Setting

Any client-specific PMP configuration has to be done using a separate function, DRV_PMP_ModeConfig. This function must be called after the client is open using DRV_PMP_Open.

Following are the client-specific configuration parameters the user can set using this function:

Configuration Parameter	Description
pmpMode	Selects the PMP mode (master or slave) to use.
intMode	Selects the interrupt mode to use.
incrementMode	Sets up address for either auto-increment or decrement mode.
endianMode	Sets Little/Big endian mode.
portSize	Specifies the data width (8-bit or 16-bit).
waitStates	Selects the different wait states.
chipSelect	Selects the Chip Select line.

Example:

```
DRV_HANDLE handle;
DRV_PMP_MODE_CONFIG config;

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
```

```
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize =  PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

// Configure the client
DRV_PMP_ModeConfig ( handle, config );
```

Example Code for Complete Operation

A code example of complete operation is provided in this section.

Description

This example code will write 100 bytes of data twice (i.e., repeat once) in the memory location starting from 0x0206, and then it will be read in the buffer, myReadBuffer. The modes selected for this transfer are:

- Non-blocking
- No Interrupt
- PMP Master Mode 2
- Address Auto-increment
- No Address/Data Lines Multiplexing
- 8-bit data

Example:

```
void main(void)
{
DRV_PMP_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       pmpStatus;
DRV_HANDLE       handle;
DRV_PMP_MODE_CONFIG config;
unsigned char myReadBuffer[300], myWriteBuffer[100];
uint32_t deviceAddress, nBytes, repeatCount, i;
uint32_t writeID, readID;
DRV_PMP_TRANSFER_STATUS writeStatus=0, readStatus=0;


// populate the PMP init configuration structure
    init.inputBuffer = PMP_INPUT_BUFFER_TTL;
    init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
    init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS;
    init.ports.readWriteStrobe = PORT_ENABLE;
    init.ports.writeEnableStrobe = PORT_ENABLE;
    init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
    init.pmpID            = PMP_ID_0;
    init.stopInIdle       = false;
    init.muxMode          = PMP_MUX_NONE;

object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

pmpStatus = DRV_PMP_Status(object);

if ( SYS_STATUS_READY != pmpStatus)
{
    // Handle error
}

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);
```

```
if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize =  PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Configure the client
DRV_PMP_ModeConfig ( handle, config );

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
   myWriteBuffer[i]=i*5+7;
}

writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes*2);


while(!((writeStatus == PMP_TRANSFER_FINISHED)&&(readStatus == PMP_TRANSFER_FINISHED)))
{
    DRV_PMP_Tasks (object);

    writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
    readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);
}

while(1);
}
```

Configuring the Library

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

Description

The configuration of the PMP driver is based on the file `drv_pmp_config.h`.

This header file contains the configuration selection for the PMP Driver. Based on the selections made, the PMP Driver may support the selected features. These configuration settings will apply to all instances of the PMP Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_PMP_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_pmp_config.h

C

```
#define DRV_PMP_CLIENTS_NUMBER 2
```

Description

PMP maximum number of clients

This definition select the maximum number of clients that the PMP driver can support at run time.

Remarks

None.

DRV_PMP_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_pmp_config.h

C

```
#define DRV_PMP_INSTANCES_NUMBER 1
```

Description

PMP hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None.

DRV_PMP_QUEUE_SIZE Macro

PMP queue size for different instances.

File

drv_pmp_config.h

C

```
#define DRV_PMP_QUEUE_SIZE 8
```

Description

PMP queue size

The PMP queue size for a driver instances should be placed here. If more than one driver instance of PMP is present, then all takes the same queue size.

Remarks

All the transfers (Read/Write) first gets queued and gets completed sequentially when Task API is called in a loop. Therefore, the minimum value of this index should be 1.

Building the Library

This section lists the files that are available in the PMP Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/pmp`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_pmp.h`	This file provides the interface definitions of the PMP driver

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_pmp_dynamic.c`	This file contains the core implementation of the PMP driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The PMP Driver Library depends on the following modules:

- PMP Peripheral Library
- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Initialize	Initializes the PMP driver. **Implementation:** Static/Dynamic
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_PMP_Status	Provides the current status of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_PMP_TimingSet	Sets PMP timing parameters. **Implementation:** Static

b) Client Interaction Functions

	Name	Description
	DRV_PMP_ClientStatus	Gets the current client-specific status of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Close	Closes an opened instance of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_ModeConfig	Configures the PMP modes. **Implementation:** Static/Dynamic
	DRV_PMP_Open	Opens the specified PMP driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_PMP_Read	Read the data from external device. **Implementation:** Static/Dynamic
	DRV_PMP_Write	Transfers the data from the MCU to the external device. **Implementation:** Static/Dynamic

c) Client Transfer Functions

	Name	Description
	DRV_PMP_TransferStatus	Returns the transfer status. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_PMP_INDEX_COUNT	Number of valid PMP driver indices.
	DRV_PMP_CHIPX_STROBE_MODE	PMP writeEnable/ReadWrite strobes.
	DRV_PMP_CLIENT_STATUS	PMP client status definitions.
	DRV_PMP_ENDIAN_MODE	PMP Endian modes.
	DRV_PMP_INDEX	PMP driver index definitions.
	DRV_PMP_INIT	Defines the PMP driver initialization data.
	DRV_PMP_MODE_CONFIG	PMP modes configuration.
	DRV_PMP_POLARITY_OBJECT	PMP polarity object.
	DRV_PMP_PORT_CONTROL	PMP port enable/disable definitions.
	DRV_PMP_PORTS	PMP port configuration.
	DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	_DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_TRANSFER_STATUS	Defines the PMP transfer status.
	DRV_PMP_WAIT_STATES	PMP wait states object.
	MAX_NONBUFFERED_BYTE_COUNT	After this number the PMP transfer should be polled to guarantee data transfer
	DRV_PMP_TRANSFER_TYPE	This is type DRV_PMP_TRANSFER_TYPE.
	PMP_QUEUE_ELEMENT_OBJECT	Defines the structure required for maintaining the queue element.

Description

This section describes the Application Programming Interface (API) functions of the PMP Driver.

Refer to each section for a detailed description.

a) System Functions

DRV_PMP_Deinitialize Function

Deinitializes the specified instance of the PMP driver module.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Deinitialize(const SYS_MODULE_OBJ pmpDriverObject);
```

Returns

None.

Description

This function deinitializes the specified instance of the PMP driver module, disabling its operation (and any hardware). All internal data is invalidated.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_PMP_Status operation. The system has to use DRV_PMP_Status to find out when the module is in the ready state.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     //  Returned from DRV_PMP_Initialize
SYS_STATUS          status;

DRV_PMP_Deinitialize(pmpDriverObject);
```

```
status = DRV_PMP_Status(pmpDriverObject);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Function

void DRV_PMP_Deinitialize (SYS_MODULE_OBJ pmpDriverObject)

DRV_PMP_Initialize Function

Initializes the PMP driver.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
SYS_MODULE_OBJ DRV_PMP_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, it returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to DRV_PMP_Reinitialize, DRV_PMP_Deinitialize, DRV_PMP_Tasks and DRV_PMP_Status routines.

Description

This function initializes the PMP driver, making it ready for clients to open and use it.

Remarks

This function must be called before any other PMP function is called.

This function should only be called once during system initialization unless DRV_PMP_Deinitialize is called to deinitialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_PMP_Status operation. The system must use DRV_PMP_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
DRV_PMP_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the initialization structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID            = PMP_ID_0;
init.stopInIdle       = false;
init.muxMode          = PMP_MUX_NONE;

// Do something
```

```
objectHandle = DRV_PMP_Initialize(DRV_PMP_INDEX_0,  (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver

Function

SYS_MODULE_OBJ DRV_PMP_Initialize(const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT * const init)

DRV_PMP_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Reinitialize(const SYS_MODULE_OBJ pmpDriverObject, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function reinitializes the driver and refreshes any associated hardware settings using the specified initialization data, but it will not interrupt any ongoing operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the DRV_PMP_Status operation. The system must use DRV_PMP_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_PMP_INIT    init;
SYS_MODULE_OBJ  pmpDriverObject;
SYS_STATUS  pmpStatus;

// Populate the initialization structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID            = PMP_ID_0;
```

```
init.stopInIdle       = false;
init.muxMode          = PMP_MUX_NONE;

DRV_PMP_Reinitialize(pmpDriverObject, (SYS_MODULE_INIT*)&init);

pmpStatus = DRV_PMP_Status(pmpDriverObject);
if (SYS_STATUS_BUSY == pmpStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= pmpStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Function

void DRV_PMP_Reinitialize (SYS_MODULE_OBJ pmpDriverObject,
const SYS_MODULE_INIT * const init)

init - Pointer to the initialization data structure

DRV_PMP_Status Function

Provides the current status of the PMP driver module.

Implementation: Dynamic

File

drv_pmp.h

C

```
SYS_STATUS DRV_PMP_Status(const SYS_MODULE_OBJ pmpDriverObject);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the PMP driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     // Returned from DRV_PMP_Initialize
SYS_STATUS          status;

status = DRV_PMP_Status(pmpDriverObject);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize routine

Function

SYS_STATUS DRV_PMP_Status (SYS_MODULE_OBJ pmpDriverObject)

DRV_PMP_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Tasks(SYS_MODULE_OBJ pmpDriverObject);
```

Returns

None.

Description

This function is used to maintain the queue and execute the tasks stored in the queue. It resides in the ISR of the PMP for interrupt-driven implementations.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_PMP_Initialize function must have been called for the specified PMP driver instance.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     // Returned from DRV_PMP_Initialize

while (true)
{
    DRV_PMP_Tasks (pmpDriverObject);

    // Do other tasks
}
```

Parameters

Parameters	Description
pmpDriverObject	Object handle for the specified driver instance (returned from DRV_PMP_Initialize)

Function

void DRV_PMP_Tasks (SYS_MODULE_OBJ pmpDriverObject);

DRV_PMP_TimingSet Function

Sets PMP timing parameters.

Implementation: Static

File

drv_pmp.h

C

```
void DRV_PMP_TimingSet(PMP_DATA_WAIT_STATES dataWait, PMP_STROBE_WAIT_STATES strobeWait,
PMP_DATA_HOLD_STATES dataHold);
```

Returns

None.

Description

This function sets the PMP timing parameters.

Remarks

None.

Preconditions

The DRV_PMP_Initialize function must have been called.

Example

```
DRV_PMP0_TimingSet(PMP_DATA_WAIT_THREE,PMP_STROBE_WAIT_6,PMP_DATA_HOLD_4);
```

Parameters

Parameters	Description
dataWait	Data setup to read/write strobe wait states
strobeWait	Read/write strobe wait states
dataHold	Data hold time after read/write strobe wait states

Function

void DRV_PMP_TimingSet(
PMP_DATA_WAIT_STATES dataWait,
PMP_STROBE_WAIT_STATES strobeWait,
PMP_DATA_HOLD_STATES dataHold
)

b) Client Interaction Functions

DRV_PMP_ClientStatus Function

Gets the current client-specific status of the PMP driver.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_PMP_CLIENT_STATUS DRV_PMP_ClientStatus(DRV_HANDLE hClient);
```

Returns

A DRV_PMP_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the PMP driver associated with the specified handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_PMP_Initialize routine must have been called.

DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
DRV_PMP_CLIENT_STATUS pmpClientStatus;

pmpClientStatus = DRV_PMP_ClientStatus(hClient);
if(DRV_PMP_CLIENT_STATUS_ERROR >= pmpClientStatus)
{
    // Handle the error
}
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine

Function

DRV_PMP_CLIENT_STATUS DRV_PMP_ClientStatus (DRV_HANDLE hClient)

DRV_PMP_Close Function

Closes an opened instance of the PMP driver.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Close(const DRV_HANDLE hClient);
```

Returns

None

Description

This function closes an opened instance of the PMP driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_PMP_Open before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the DRV_PMP_Status operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_PMP_Initialize routine must have been called for the specified PMP driver instance.

DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open

DRV_PMP_Close(hClient);
```

Parameters

Parameters	Description
hClient	A valid open instance handle, returned from the driver's open routine

Function

void DRV_PMP_Close (DRV_HANDLE hClient)

DRV_PMP_ModeConfig Function

Configures the PMP modes.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_ModeConfig(DRV_HANDLE hClient, DRV_PMP_MODE_CONFIG config);
```

Returns

None.

Description

This function configures the modes for client in which it wants to operate. Different master-slave modes, 8/16 data bits selection, address increment/decrement, interrupt mode, wait states, etc., can be configured through this function.

Remarks

This function will NEVER block waiting for hardware. If this API is called more than once for a particular client handle, previous config setting of that client will be overwritten.

Preconditions

Function DRV_PMP_Initialize must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE  hClient;
DRV_PMP_MODE_CONFIG config;

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT;
config.portSize =  PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

DRV_PMP_ModeConfig ( hClient, config );
```

Parameters

Parameters	Description
hClient	Client handle obtained from DRV_PMP_Open API
config	Structure which will have all the required PMP modes configuration

Function

void DRV_PMP_ModeConfig (DRV_HANDLE hClient,
DRV_PMP_MODE_CONFIG config)

DRV_PMP_Open Function

Opens the specified PMP driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_HANDLE DRV_PMP_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This function opens the specified PMP driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_PMP_Close routine is called.

This function will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the DRV_PMP_ClientStatus operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  hClient;

hClient = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == hClient)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_PMP_Open (const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT intent)

DRV_PMP_Read Function

Read the data from external device.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
PMP_QUEUE_ELEMENT_OBJECT* DRV_PMP_Read(DRV_HANDLE hClient, uint32_t address, uint16_t* buffer, uint32_t
nBytes);
```

Returns

Returns the position number of the queue, where the data element was stored. Returns '0' when there is no place in the queue to store the data.

Description

This function reads the given number of data bytes from the given address of the external device to the MCU buffer through the selected PMP instance. This function should be used for all the master and slave modes. Proper configuration should be done using DRV_PMP_ModeConfig before calling this function.

Preconditions

The DRV_PMP_Initialize routine must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle. DRV_PMP_ModeConfig must have been called to configure the desired mode

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
uint32_t deviceAddress;
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t transferID;

transferID = DRV_PMP_Read ( hClient, deviceAddress, &myBuffer, nBytes);
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device from where data has to be read. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode i.e. addressable buffer slave mode, this parameter should be the buffer number to be used.
buffer	Pointer to the buffer into which the data read through the PMP instance will be placed. Even if only one word has to be transferred, pointer should be used.
nBytes	Number of bytes that need to be read through the PMP instance

Function

uint32_t DRV_PMP_Read (DRV_HANDLE hClient,

uint32_t address,

unsigned char* buffer,

uint32_t nBytes)

DRV_PMP_Write Function

Transfers the data from the MCU to the external device.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
PMP_QUEUE_ELEMENT_OBJECT* DRV_PMP_Write(DRV_HANDLE* hClient, bool address, uint32_t * buffer, uint32_t
nBytes, uint32_t repeatCount);
```

Returns

Returns a 32-bit ID with which status of the transfer can be checked later. Returns '0' when there is no place in the queue to store the data.

Description

This function transfer the given number of data bytes from the MCU buffer location to the defined address of the external device through the selected PMP instance. It repeats the operation n (=repeatCount) number of times as well. This function should be used for all the master and slave modes. Proper configuration should be done using DRV_PMP_ModeConfig before calling this function.

Preconditions

The DRV_PMP_Initialize routine must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle. DRV_PMP_ModeConfig must have been called to configure the desired mode.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
uint32_t deviceAddress;
```

```
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t repeatCount;
uint32_t transferID;

transferID = DRV_PMP_MasterWrite ( hClient, deviceAddress, &myBuffer, nBytes, repeatCount);
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device where data has to be written. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode (i.e., addressable buffer slave mode), this parameter should be the buffer number to be used.
buffer	Pointer to MCU Buffer from which the data will be written through the PMP instance. even if only one word has to be transferred, pointer should be used.
nBytes	Total number of bytes that need to be written through the PMP instance
repeatCount	Number of times the data set (nBytes of data) to be repeatedly written. This value should be 0 if user does not want any repetition. If repeatCount is greater than 0, then after writing every nBytes of data, the buffer starts pointing to its first element. Ideally, PMP Address should be in auto increment/decrement mode for repeatCount greater than 0.

Function

uint32_t DRV_PMP_Write (DRV_HANDLE hClient,
uint32_t address,
unsigned char* buffer,
uint32_t nBytes,
uint32_t repeatCount)

c) Client Transfer Functions

DRV_PMP_TransferStatus Function

Returns the transfer status.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_PMP_TRANSFER_STATUS DRV_PMP_TransferStatus(PMP_QUEUE_ELEMENT_OBJECT* queueObject);
```

Returns

A DRV_PMP_TRANSFER_STATUS value describing the current status of the transfer.

Description

This function returns the status of a particular transfer whose ID has been specified as input.

Example

```
uint32_8 seqID;
DRV_PMP_TRANSFER_STATUS transferStatus;

transferStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, seqID);
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
seqID	A valid ID returned from read/write transfer functions

Function

DRV_PMP_TRANSFER_STATUS DRV_PMP_TransferStatus(DRV_HANDLE hClient)

d) Miscellaneous Functions

e) Data Types and Constants

DRV_PMP_INDEX_COUNT Macro

Number of valid PMP driver indices.

File

drv_pmp.h

C

```
#define DRV_PMP_INDEX_COUNT _PMP_EXISTS
```

Description

PMP Driver Module Index Count

This constant identifies the number of valid PMP driver indices.

Remarks

The value of "_PMP_EXISTS" is derived from device-specific header files defined as part of the peripheral libraries.

DRV_PMP_CHIPX_STROBE_MODE Enumeration

PMP writeEnable/ReadWrite strobes.

File

drv_pmp.h

C

```
typedef enum {
  PMP_RW_STROBE_WITH_ENABLE_STROBE,
  PMP_READ_AND_WRITE_STROBES
} DRV_PMP_CHIPX_STROBE_MODE;
```

Members

Members	Description
PMP_RW_STROBE_WITH_ENABLE_STROBE	One strobe for read/write and another for enable
PMP_READ_AND_WRITE_STROBES	Separate strobes for read and write operations

Description

PMP writeEnable/ReadWrite strobes

This enumeration provides ReadWrite/WriteEnable Strobe definitions.

DRV_PMP_CLIENT_STATUS Enumeration

PMP client status definitions.

File

drv_pmp.h

C

```
typedef enum {
  DRV_PMP_CLIENT_STATUS_INVALID,
  PMP_CLIENT_STATUS_CLOSED,
  DRV_PMP_CLIENT_STATUS_OPEN
} DRV_PMP_CLIENT_STATUS;
```

Description

PMP Client Status

This enumeration provides various client status possibilities.

DRV_PMP_ENDIAN_MODE Enumeration

PMP Endian modes.

File

drv_pmp.h

C

```
typedef enum {
  LITTLE,
  BIG
} DRV_PMP_ENDIAN_MODE;
```

Members

Members	Description
LITTLE	Little Endian
BIG	Big Endian

Description

PMP Endian modes

This enumeration holds the Endian configuration options.

DRV_PMP_INDEX Enumeration

PMP driver index definitions.

File

drv_pmp.h

C

```
typedef enum {
  DRV_PMP_INDEX_0,
  DRV_PMP_INDEX_1
} DRV_PMP_INDEX;
```

Members

Members	Description
DRV_PMP_INDEX_0	First PMP instance
DRV_PMP_INDEX_1	Second PMP instance (not available for now)

Description

PMP Driver Module Index Numbers

These constants provide PMP driver index definitions.

Remarks

These values should be passed into the DRV_PMP_Initialize and DRV_PMP_Open functions to identify the driver instance in use.

DRV_PMP_INIT Structure

Defines the PMP driver initialization data.

File

drv_pmp.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
```

```
  PMP_MODULE_ID pmpID;
  bool stopInIdle;
  PMP_MUX_MODE muxMode;
  PMP_INPUT_BUFFER_TYPE inputBuffer;
  DRV_PMP_POLARITY_OBJECT polarity;
  DRV_PMP_PORTS ports;
} DRV_PMP_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	module power state info
PMP_MODULE_ID pmpID;	module PLIB ID
bool stopInIdle;	Stop in Idle enable
PMP_MUX_MODE muxMode;	MUX mode
PMP_INPUT_BUFFER_TYPE inputBuffer;	Input buffer type to be used
DRV_PMP_POLARITY_OBJECT polarity;	Polarity settings
DRV_PMP_PORTS ports;	PMP port settings

Description

PMP Driver Initialize Data

This data type defines data required to initialize or reinitialize the PMP driver.

Remarks

Not all the initialization features are available for all devices.

DRV_PMP_MODE_CONFIG Structure

PMP modes configuration.

File

drv_pmp.h

C

```
typedef struct {
  PMP_OPERATION_MODE pmpMode;
  PMP_INTERRUPT_MODE intMode;
  PMP_INCREMENT_MODE incrementMode;
  DRV_PMP_ENDIAN_MODE endianMode;
  PMP_DATA_SIZE portSize;
  DRV_PMP_WAIT_STATES waitStates;
  PMP_CHIPSELECT_FUNCTION chipSelect;
} DRV_PMP_MODE_CONFIG;
```

Members

Members	Description
PMP_OPERATION_MODE pmpMode;	PMP Usage Mode Type
PMP_INTERRUPT_MODE intMode;	Interrupt mode
PMP_INCREMENT_MODE incrementMode;	should be appropriately selected based on read/write requirements and operation mode setting */ address/buffer increment mode
DRV_PMP_ENDIAN_MODE endianMode;	it does not have any significance in PMP slave mode or 8bit data mode */ Endian modes
PMP_DATA_SIZE portSize;	Data Port Size
DRV_PMP_WAIT_STATES waitStates;	Wait states
PMP_CHIPSELECT_FUNCTION chipSelect;	use this when PLIB is fixed

Description

PMP modes configuration

This data type controls the configuration of PMP modes.

DRV_PMP_POLARITY_OBJECT Structure

PMP polarity object.

File

drv_pmp.h

C

```
typedef struct {
  PMP_POLARITY_LEVEL addressLatchPolarity;
  PMP_POLARITY_LEVEL byteEnablePolarity;
  PMP_POLARITY_LEVEL rwStrobePolarity;
  PMP_POLARITY_LEVEL writeEnableStrobePolarity;
  PMP_POLARITY_LEVEL chipselect1Polarity;
  PMP_POLARITY_LEVEL chipselect2Polarity;
} DRV_PMP_POLARITY_OBJECT;
```

Members

Members	Description
PMP_POLARITY_LEVEL addressLatchPolarity;	Address latch polarity
PMP_POLARITY_LEVEL byteEnablePolarity;	ByteEnable port polarity
PMP_POLARITY_LEVEL rwStrobePolarity;	Read/Write strobe polarity
PMP_POLARITY_LEVEL writeEnableStrobePolarity;	Write/Enable strobe polarity
PMP_POLARITY_LEVEL chipselect1Polarity;	ChipSelect-1 Polarity
PMP_POLARITY_LEVEL chipselect2Polarity;	chipSelect-2 Polarity

Description

PMP polarity object

This structure holds the polarities of different entities to be configured.

DRV_PMP_PORT_CONTROL Enumeration

PMP port enable/disable definitions.

File

drv_pmp.h

C

```
typedef enum {
  PORT_ENABLE,
  PORT_DISABLE
} DRV_PMP_PORT_CONTROL;
```

Members

Members	Description
PORT_ENABLE	Enable the given port
PORT_DISABLE	Disable the given port

Description

PMP port enable/disable.

This enumeration provides port enable/disable values.

DRV_PMP_PORTS Structure

PMP port configuration.

File

drv_pmp.h

C

```
typedef struct {
  PMP_ADDRESS_PORT addressPortsMask;
  PMP_PMBE_PORT byteEnablePort;
  DRV_PMP_PORT_CONTROL readWriteStrobe;
  DRV_PMP_PORT_CONTROL writeEnableStrobe;
```

```
} DRV_PMP_PORTS;
```

Members

Members	Description
PMP_ADDRESS_PORT addressPortsMask;	User needs to put the address lines which he wants to use in ORed fashion * Address ports
PMP_PMBE_PORT byteEnablePort;	Byte enable ports
DRV_PMP_PORT_CONTROL readWriteStrobe;	READ/WRITE Strobe PORT
DRV_PMP_PORT_CONTROL writeEnableStrobe;	WRITE/ENABLE strobe port

Description

PMP Ports

This structure holds the ports (including the address ports) to be configured by the application to function as general purpose I/O (GPIO) or part of the PMP.

DRV_PMP_QUEUE_ELEMENT_OBJ Structure

Defines the object for PMP queue element.

File

drv_pmp.h

C

```
typedef struct _DRV_PMP_QUEUE_ELEMENT_OBJ {
  struct _DRV_PMP_CLIENT_OBJ * hClient;
  uint32_t buffer;
  uint16_t* addressBuffer;
  uint32_t nTransfers;
  int32_t nRepeats;
  DRV_PMP_TRANSFER_TYPE type;
} DRV_PMP_QUEUE_ELEMENT_OBJ;
```

Members

Members	Description
struct _DRV_PMP_CLIENT_OBJ * hClient;	handle of the client object returned from open API
uint32_t buffer;	pointer to the buffer holding the transmitted data
uint16_t* addressBuffer;	pointer to the buffer holding the transmitted data
uint32_t nTransfers;	number of bytes to be transferred
int32_t nRepeats;	number of times the data set has to be transferred repeatedly
DRV_PMP_TRANSFER_TYPE type;	PMP Read or Write

Description

PMP Driver Queue Element Object

This defines the object structure for each queue element of PMP. This object gets created for every Read/Write operations APIs.

Remarks

None

DRV_PMP_TRANSFER_STATUS Enumeration

Defines the PMP transfer status.

File

drv_pmp.h

C

```
typedef enum {
  MASTER_8BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_8_BITS,
  MASTER_16BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_16_BITS,
  MASTER_8BIT_BUFFER_IN_PROGRESS,
  MASTER_16BIT_BUFFER_IN_PROGRESS,
  MASTER_8BIT_TRANSFER_CONTINUE,
  MASTER_8BIT_BUFFER_CONTINUE,
```

```
  QUEUED_BUT_PMP_TRANSFER_NOT_STARTED,
  PMP_TRANSFER_FINISHED
} DRV_PMP_TRANSFER_STATUS;
```

Description

Queue Element Transfer Status

This enumeration defines the PMP transfer status.

DRV_PMP_WAIT_STATES Structure

PMP wait states object.

File

drv_pmp.h

C

```
typedef struct {
  PMP_DATA_HOLD_STATES dataHoldWait;
  PMP_STROBE_WAIT_STATES strobeWait;
  PMP_DATA_WAIT_STATES dataWait;
} DRV_PMP_WAIT_STATES;
```

Members

Members	Description
PMP_DATA_HOLD_STATES dataHoldWait;	data hold wait states
PMP_STROBE_WAIT_STATES strobeWait;	read/write strobe wait states
PMP_DATA_WAIT_STATES dataWait;	data wait strobe wait sates

Description

PMP wait states object

This structure holds the different wait states to be configured. Refer to the PMP PLIB help document for the possible values and meaning of the different wait states.

MAX_NONBUFFERED_BYTE_COUNT Macro

File

drv_pmp.h

C

```
#define MAX_NONBUFFERED_BYTE_COUNT 4 /*************************************************************************
                                      After this number the PMP transfer should be polled to guarantee data
                                      transfer
                                      *************************************************************************

*/
```

Description

After this number the PMP transfer should be polled to guarantee data transfer

DRV_PMP_TRANSFER_TYPE Enumeration

File

drv_pmp.h

C

```
typedef enum {
  ADDRESS,
  READ,
  WRITE,
  BUFFERED_WRITE
} DRV_PMP_TRANSFER_TYPE;
```

Members

Members	Description
ADDRESS	PMP Address needs to be updated
READ	PMP Read Transfer
WRITE	PMP Write Transfer
BUFFERED_WRITE	PMP Array Write Transfer

Description

This is type DRV_PMP_TRANSFER_TYPE.

PMP_QUEUE_ELEMENT_OBJECT Structure

Defines the structure required for maintaining the queue element.

File

drv_pmp.h

C

```
typedef struct {
  DRV_PMP_QUEUE_ELEMENT_OBJ data;
  DRV_PMP_TRANSFER_STATUS eTransferStatus;
  uint32_t nTransfersDone;
} PMP_QUEUE_ELEMENT_OBJECT;
```

Members

Members	Description
DRV_PMP_QUEUE_ELEMENT_OBJ data;	The PMP Q Element
DRV_PMP_TRANSFER_STATUS eTransferStatus;	Flag to indicate that the element is in use
uint32_t nTransfersDone;	sequence id

Description

Queue Element Object

This defines the structure required for maintaining the queue element.

Remarks

None

Files

Files

Name	Description
drv_pmp.h	Parallel Master Port (PMP) device driver interface file.
drv_pmp_config.h	PMP driver configuration definitions template

Description

This section lists the source and header files used by the PMP Driver Library.

drv_pmp.h

Parallel Master Port (PMP) device driver interface file.

Enumerations

	Name	Description
	DRV_PMP_CHIPX_STROBE_MODE	PMP writeEnable/ReadWrite strobes.
	DRV_PMP_CLIENT_STATUS	PMP client status definitions.
	DRV_PMP_ENDIAN_MODE	PMP Endian modes.
	DRV_PMP_INDEX	PMP driver index definitions.

	DRV_PMP_PORT_CONTROL	PMP port enable/disable definitions.
	DRV_PMP_TRANSFER_STATUS	Defines the PMP transfer status.
	DRV_PMP_TRANSFER_TYPE	This is type DRV_PMP_TRANSFER_TYPE.

Functions

	Name	Description
	DRV_PMP_ClientStatus	Gets the current client-specific status of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Close	Closes an opened instance of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Initialize	Initializes the PMP driver. **Implementation:** Static/Dynamic
	DRV_PMP_ModeConfig	Configures the PMP modes. **Implementation:** Static/Dynamic
	DRV_PMP_Open	Opens the specified PMP driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_PMP_Read	Read the data from external device. **Implementation:** Static/Dynamic
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_PMP_Status	Provides the current status of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_PMP_TimingSet	Sets PMP timing parameters. **Implementation:** Static
	DRV_PMP_TransferStatus	Returns the transfer status. **Implementation:** Dynamic
	DRV_PMP_Write	Transfers the data from the MCU to the external device. **Implementation:** Static/Dynamic

Macros

	Name	Description
	DRV_PMP_INDEX_COUNT	Number of valid PMP driver indices.
	MAX_NONBUFFERED_BYTE_COUNT	After this number the PMP transfer should be polled to guarantee data transfer

Structures

	Name	Description
	_DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_INIT	Defines the PMP driver initialization data.
	DRV_PMP_MODE_CONFIG	PMP modes configuration.
	DRV_PMP_POLARITY_OBJECT	PMP polarity object.
	DRV_PMP_PORTS	PMP port configuration.
	DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_WAIT_STATES	PMP wait states object.
	PMP_QUEUE_ELEMENT_OBJECT	Defines the structure required for maintaining the queue element.

Description

PMP Device Driver Interface

The PMP device driver provides a simple interface to manage the Parallel Master and Slave ports. This file defines the interface definitions and prototypes for the PMP driver.

File Name

drv_pmp.h

Company

Microchip Technology Inc.

drv_pmp_config.h

PMP driver configuration definitions template

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

Description

PMP Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_pmp_config_template.h

Company

Microchip Technology Inc.

RTCC Driver Library

This section describes the RTCC Driver Library.

Introduction

The Real-Time Clock Calendar (RTCC) Static Driver provides a high-level interface to manage the RTCC module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Starting/Stopping the RTCC
- Status functions to yield the date/time
- Status functions to yield the alarm date/time
- Clock output control

Library Interface

System Interaction Functions

	Name	Description
	DRV_RTCC_AlarmDateGet	Gets the Alarm Date of the RTCC. **Implementation:** Static
	DRV_RTCC_AlarmTimeGet	Gets the Alarm Time of the RTCC. **Implementation:** Static
	DRV_RTCC_ClockOutput	Enables Clock Output for the RTCC. **Implementation:** Static
	DRV_RTCC_DateGet	Gets the Date of the RTCC. **Implementation:** Static
	DRV_RTCC_Initialize	Initializes the RTCC instance for the specified driver index. **Implementation:** Static
	DRV_RTCC_Start	Starts the RTCC. **Implementation:** Static

	DRV_RTCC_Stop	Stops the RTCC. **Implementation:** Static
	DRV_RTCC_TimeGet	Gets the time of the RTCC. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the RTCC Driver Library.

System Interaction Functions

DRV_RTCC_AlarmDateGet Function

Gets the Alarm Date of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_AlarmDateGet();
```

Returns

uint32_t alarm date value

Description

This routine gets the RTCC alarm date.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_AlarmDateGet(void)

DRV_RTCC_AlarmTimeGet Function

Gets the Alarm Time of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_AlarmTimeGet();
```

Returns

uint32_t alarm time value

Description

This routine gets the RTCC alarm time.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_AlarmTimeGet(void)

DRV_RTCC_ClockOutput Function

Enables Clock Output for the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_ClockOutput();
```

Returns

None.

Description

This routine enables the clock output for the RTCC

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_ClockOutput(void)

DRV_RTCC_DateGet Function

Gets the Date of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_DateGet();
```

Returns

uint32_t date value

Description

This routine gets the RTCC date.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_DateGet(void)

DRV_RTCC_Initialize Function

Initializes the RTCC instance for the specified driver index.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Initialize();
```

Returns

None.

Description

This routine initializes the RTCC driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This routine must be called before any other RTCC routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_RTCC_Initialize(void)

DRV_RTCC_Start Function

Starts the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Start();
```

Returns

None.

Description

This routine starts the RTCC, making it ready for clients to use it.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_Start(void)

DRV_RTCC_Stop Function

Stops the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Stop();
```

Returns

None.

Description

This routine stops the RTCC.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_Stop(void)

DRV_RTCC_TimeGet Function

Gets the time of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_TimeGet();
```

Returns

uint32_t time value

Description

This routine gets the RTCC time.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_TimeGet(void)

Secure Digital (SD) Card Driver Library

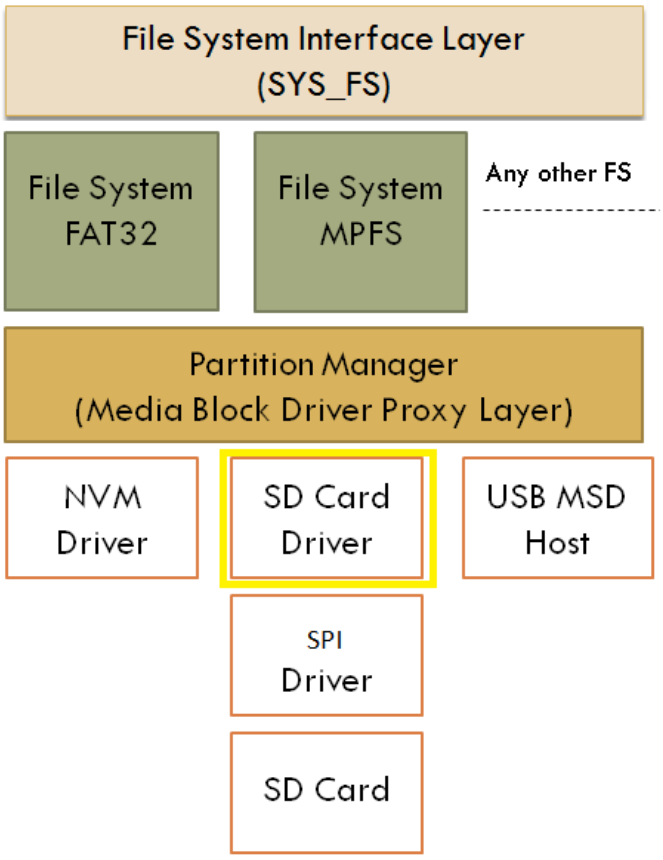
This section describes the Secure Digital (SD) Card Driver Library.

Introduction

The SD Card driver provides the necessary interfaces to interact with an SD card. It provides the necessary abstraction for the higher layer.

Description

A SD Card is a non-volatile memory (Flash memory) card designed to provide high-capacity memory in a small size. Its applications include digital video camcorders, digital cameras, handheld computers, audio players, and mobile phones.



Using the Library

This topic describes the basic architecture of the SD Card Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_sdcard.h`

The interface to the SD Card Driver library is defined in the `drv_sdcard.h` header file. This file is included by the `drv.h` file. Any C language source (`.c`) file that uses the SD Card Driver library should include `drv.h`.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

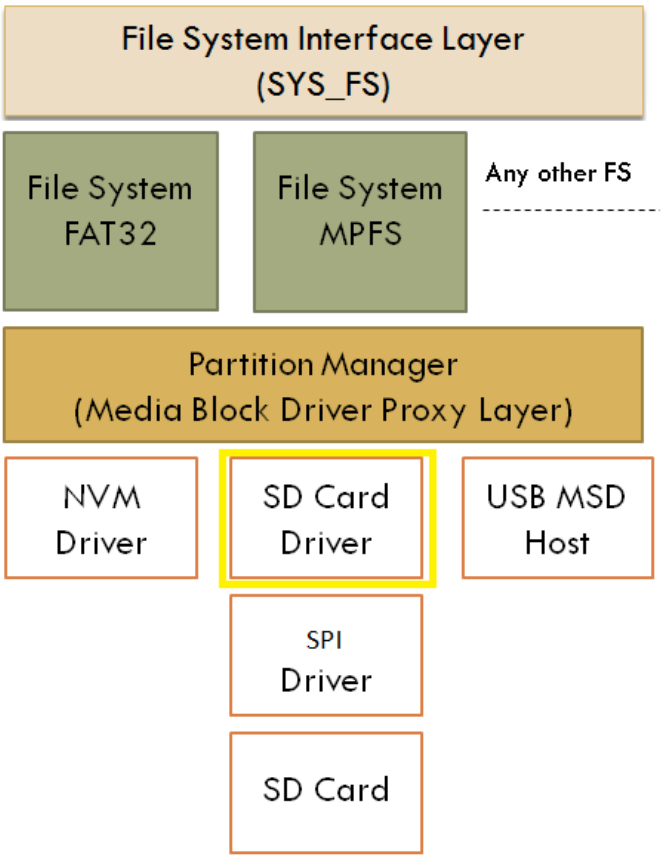
Abstraction Model

This library provides a low-level abstraction of the SD Card Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SD Card driver comes in the layer below the Partition Manager in the MPLAB Harmony file system architecture and it uses the SPI Driver to interact with the SD card.

SD Card Driver Software Abstraction Block Diagram



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SD Card module.

Library Interface Section	Description
System Level Functions	Includes functions for initialize the module.
Client Level Functions	Functions to open and close a client.
Operation Functions	Functions for read and write operations
Module Information Functions	Functions for information about the module.
Version Information Functions	Functions to get the software version.

How the Library Works

This section describes how the SD Card Driver Library operates.

Description

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

The library provides interfaces that support:

- Driver Initialization Functionality
- Client Block Data Functionality
- Client Access Functionality

SD Card Driver Initialization

This section provides information for system initialization and reinitialization.

Description

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the SD Card module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_SDCARD_INIT or by using initialization overrides) that are supported by the specific SD Card device hardware:

- SPI Peripheral ID: Identifies the SPI Peripheral ID to be used for the SD Card Driver
- SPI Index: SPI Driver Index
- SD Card frequency: SD Card communication speed
- SPI Clock source: Peripheral clock used by the SPI
- Write-Protect Port: Port used to check if the SD Card is write protected
- Write-Protect Pin: Pin used to check if the SD Card is write protected
- Chip Select Port: Port used for the SPI Chip Select
- Chip Select Pin: Pin used for the SPI Chip Select

The DRV_SDCARD_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_SDCARD_Deinitialize, DRV_SDCARD_Status, and DRV_SDCARD_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

Example:

```
const DRV_SDCARD_INIT drvSDCardInit =
{
.spiId = SPI_ID_2,
.spiIndex = 0,
.sdcardSpeedHz = 20000000,
.spiClk = CLK_BUS_PERIPHERAL_2,
.writeProtectPort = PORT_CHANNEL_F,
.writeProtectBitPosition = PORTS_BIT_POS_1,
.chipSelectPort = PORT_CHANNEL_B,
.chipSelectBitPosition = PORTS_BIT_POS_14,
};

void SYS_Initialize (void *data)
{
.
.
sysObj.drvSDCard = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0,(SYS_MODULE_INIT *)&drvSDCardInit);
.
.
}
```

Tasks Routine

The system will call DRV_SDCARD_Tasks, from system task service to maintain the driver's state machine.

Client Access Operation

This section provides information for general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_SDCARD_Open function. This provides the configuration required to open the SD Card instance for operation. If the driver is deinitialized using the function DRV_SDCARD_Deinitialize, the application must call the DRV_SDCARD_Open function again to set up the instance of the SDCARD.

For the various options available for I/O INTENT please refer to Data Types and Constants in the Library Interface section.

Example:

```
DRV_HANDLE handle;
handle = DRV_SDCARD_Open(DRV_SDCARD_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
```

```
if (DRV_HANDLE_INVALID == handle)
{
// Unable to open the driver
}
```

Client Block Data Operation

This topic provides information on client block data operation.

Description

The SDCARD Driver provides a block interface to access the SD Card. The interface provides functionality to read from and write to the SD Card.

Reading Data from the SD Card:

The following steps outline the sequence to be followed for reading data from the SD Card:

1. The system should have completed necessary initialization and DRV_SDCARD_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Invoke the DRV_SDCARD_Read function and pass the pointer where the data is to be stored, block start address and the number of blocks of data to be read.
4. The client should validate the command handle returned by the DRV_SDCARD_Read function. DRV_SDCARD_COMMAND_HANDLE_INVALID value indicates that an error has occurred which the client needs to handle.
5. If the request was successful then the client can check the status of the request by invoking the DRV_SDCARD_CommandStatus and passing the command handle returned by the read request. Alternately the client could use the event handler for notifications from the driver.
6. The client will be able to close itself by calling the DRV_SDCARD_Close.

Example:

```
// This code shows how to read data from the SD Card
DRV_HANDLE sdcardHandle;
DRV_SDCARD_COMMAND_HANDLE sdcardCommandHandle;
DRV_SDCARD_COMMAND_STATUS commandStatus;
uint8_t readBuf[512];
uint32_t blockAddress;
uint32_t nBlocks;

/* Initialize the block start address and the number of blocks to be read */
blockAddress = 0;
nBlocks = 1;

DRV_SDCARD_Read(sdcardHandle, &sdcardCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);
if(DRV_SDCARD_COMMAND_HANDLE_INVALID == sdcardCommandHandle)
{
    /* Failed to queue the read request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_SDCARD_CommandStatus(sdcardHandle, sdcardCommandHandle);
if(DRV_SDCARD_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_SDCARD_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Writing Data to the SD Card:

The following steps outline the sequence to be followed for writing data to the SD Card:

1. The system should have completed necessary initialization and DRV_SDCARD_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Invoke the DRV_SDCARD_Write function and pass the pointer to the data to be written, block start address and the number of blocks of data to be written.
4. The client should validate the command handle returned by the DRV_SDCARD_Write function. DRV_SDCARD_COMMAND_HANDLE_INVALID value indicates that an error has occurred which the client needs to handle.
5. If the request was successful then the client can check the status of the request by invoking the DRV_SDCARD_CommandStatus and passing the command handle returned by the write request. Alternately, the client could use the event handler for notifications from the driver.

6. The client will be able to close itself by calling the DRV_SDCARD_Close.

Example:

```
// This code shows how to write data to the SD Card
DRV_HANDLE sdcardHandle;
DRV_SDCARD_COMMAND_HANDLE sdcardCommandHandle;
DRV_SDCARD_COMMAND_STATUS commandStatus;
uint8_t writeBuf[512];
uint32_t blockAddress;
uint32_t nBlocks;

/* Initialize the block start address and the number of blocks to be written */
blockAddress = 0;
nBlocks = 1;
/* Populate writeBuf with the data to be written */

DRV_SDCARD_Write(sdcardHandle, &sdcardCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);
if(DRV_SDCARD_COMMAND_HANDLE_INVALID == sdcardCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_SDCARD_CommandStatus(sdcardHandle, sdcardCommandHandle);
if(DRV_SDCARD_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_SDCARD_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Configuring the Library

Macros

	Name	Description
	DRV_SDCARD_CLIENTS_NUMBER	Selects the miximum number of clients
	DRV_SDCARD_INDEX_MAX	SD Card Static Index selection
	DRV_SDCARD_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver
	DRV_SDCARD_POWER_STATE	Defines an override of the power state of the SD Card driver.
	DRV_SDCARD_SYS_FS_REGISTER	Register to use with the File system
	DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK	Enable SD Card write protect check.

Description

The configuration of the SD Card Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SD Card Driver. Based on the selections made, the SD Card Driver may support the selected features. These configuration settings will apply to all instances of the SD Card.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_SDCARD_CLIENTS_NUMBER Macro

Selects the miximum number of clients

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_CLIENTS_NUMBER 1
```

Description

SD Card Maximum Number of Clients

This definition select the maximum number of clients that the SD Card driver can support at run time. Not defining it means using a single client.

Remarks

None.

DRV_SDCARD_INDEX_MAX Macro

SD Card Static Index selection

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_INDEX_MAX 1
```

Description

SD Card Static Index Selection

SD Card Static Index selection for the driver object reference

Remarks

This index is required to make a reference to the driver object

DRV_SDCARD_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_INSTANCES_NUMBER 1
```

Description

SD Card hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None

DRV_SDCARD_POWER_STATE Macro

Defines an override of the power state of the SD Card driver.

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

SD Card power state configuration

Defines an override of the power state of the SD Card driver.

Remarks

This feature may not be available in the device or the SD Card module selected.

DRV_SDCARD_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_SYS_FS_REGISTER
```

Description

SDCARD Driver Register with File System

Specifying this macro enables the SDCARD driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the SDCARD driver is to be used with the File System.

DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK Macro

Enable SD Card write protect check.

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK
```

Description

SDCARD Driver Enable Write Protect Check

Specifying this macro enables the SDCARD driver to check whether the SD card is write protected.

Remarks

None

Building the Library

This section lists the files that are available in the SD Card Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/sdcard`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_sdcard.h`	This file provides the interface definitions of the SD Card driver

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_sdcard.c`	This file contains the core implementation of the SD Card driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SD Card Driver Library depends on the following modules:

- SPI Driver Library
- Clock System Service Library
- Interrupt System Service Library
- Ports System Service Library
- Timer System Service Library
- Timer Driver Library

Library Interface

a) System Level Functions

	Name	Description
	DRV_SDCARD_Initialize	Initializes the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_SDCARD_Status	Provides the current status of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SDCARD_Read	Reads blocks of data from the specified block address of the SD Card.
	DRV_SDCARD_Write	Writes blocks of data starting at the specified address of the SD Card.
	DRV_SDCARD_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

c) Status Functions

	Name	Description
	DRV_SDCARD_IsAttached	Returns the physical attach status of the SD Card.
	DRV_SDCARD_IsWriteProtected	Returns the write protect status of the SDCARD.
	DRV_SDCARD_CommandStatus	Gets the current status of the command.
	DRV_SDCARD_GeometryGet	Returns the geometry of the device.

d) Data Types and Constants

	Name	Description
	DRV_SDCARD_INDEX_0	SD Card driver index definitions
	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	_DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	SDCARD_DETECTION_LOGIC	Defines the different system events
	SDCARD_MAX_LIMIT	Maximum allowed SD card instances
	DRV_SDCARD_INDEX_1	This is macro DRV_SDCARD_INDEX_1.
	DRV_SDCARD_INDEX_2	This is macro DRV_SDCARD_INDEX_2.
	DRV_SDCARD_INDEX_3	This is macro DRV_SDCARD_INDEX_3.

	DRV_SDCARD_COMMAND_HANDLE_INVALID	SDCARD Driver's Invalid Command Handle.
	DRV_SDCARD_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SDCARD_COMMAND_STATUS	Identifies the possible events that can result from a request.
	DRV_SDCARD_EVENT	Identifies the possible events that can result from a request.
	DRV_SDCARD_EVENT_HANDLER	Pointer to a SDCARDDriver Event handler function

Description

This section describes the Application Programming Interface (API) functions of the SD Card Driver.

Refer to each section for a detailed description.

a) System Level Functions

DRV_SDCARD_Initialize Function

Initializes the SD Card driver.

Implementation: Dynamic

File

drv_sdcard.h

C

```
SYS_MODULE_OBJ DRV_SDCARD_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT *const init);
```

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the SD Card driver, making it ready for clients to open and use the driver.

Remarks

This routine must be called before any other SD Card routine is called.

This routine should only be called once during system initialization unless DRV_SDCARD_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_SDCARD_Status operation. The system must use DRV_SDCARD_Status to find out when the driver is in the ready state.

Preconditions

None.

Example

```
DRV_SDCARD_INIT     init;
SYS_MODULE_OBJ      objectHandle;

// Populate the SD Card initialization structure

objectHandle = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SDCARD_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init
);

DRV_SDCARD_Deinitialize Function

Deinitializes the specified instance of the SD Card driver module.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SD Card driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_SDCARD_Status operation. The system has to use DRV_SDCARD_Status to check if the de-initialization is complete.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      objectHandle;     //  Returned from DRV_SDCARD_Initialize
SYS_STATUS          status;

DRV_SDCARD_Deinitialize(objectHandle);

status = DRV_SDCARD_Status(objectHandle);
if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine.

Function

void DRV_SDCARD_Deinitialize
(
SYS_MODULE_OBJ object
);

DRV_SDCARD_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This routine reinitializes the driver and refreshes any associated hardware settings using the given initialization data, but it will not interrupt any ongoing operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_SDCARD_Status operation. The system must use DRV_SDCARD_Status to find out when the driver is in the ready state.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_SDCARD_INIT     init;
SYS_MODULE_OBJ      objectHandle;  //  Returned from DRV_SDCARD_Initialize

// Update the required fields of the SD Card initialization structure

DRV_SDCARD_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&init);
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine
init	Pointer to the initialization data structure

Function

void DRV_SDCARD_Reinitialize

(

SYS_MODULE_OBJ object,

const SYS_MODULE_INIT * const init

);

DRV_SDCARD_Status Function

Provides the current status of the SD Card driver module.

Implementation: Dynamic

File

drv_sdcard.h

C

```
SYS_STATUS DRV_SDCARD_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Note Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Description

This routine provides the current status of the SD Card driver module.

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, then a previous operation has not yet completed. If the status operation returns SYS_STATUS_READY, then it indicates that all previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SDCARD_Initialize
SYS_STATUS          status;

status = DRV_SDCARD_Status(object);

if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine

Function

SYS_STATUS DRV_SDCARD_Status

(

SYS_MODULE_OBJ object

);

DRV_SDCARD_Tasks Function

Maintains the driver's state machine.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This routine is used to maintain the driver's internal state machine.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SDCARD driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SDCARD_Initialize

while (true)
{
    DRV_SDCARD_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SDCARD_Initialize)

Function

void DRV_SDCARD_Tasks

(

SYS_MODULE_OBJ object

);

b) Client Level Functions

DRV_SDCARD_Close Function

Closes an opened-instance of the SD Card driver.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Close(DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the SD Card driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SDCARD_Open before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the DRV_SDCARD_Status operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SD Card driver instance.

DRV_SDCARD_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SDCARD_Open

DRV_SDCARD_Close (handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SDCARD_Close
(
DRV_HANDLE handle
);

DRV_SDCARD_Open Function

Opens the specified SD Card driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sdcard.h

C

```
DRV_HANDLE DRV_SDCARD_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This routine opens the specified SD Card driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_SDCARD_Close routine is called.

This routine will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, then other client-level operations may block waiting on hardware until they are complete.

If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_SDCARD_Open (DRV_SDCARD_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SDCARD_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent

);

DRV_SDCARD_Read Function

Reads blocks of data from the specified block address of the SD Card.

File

drv_sdcard.h

C

```
void DRV_SDCARD_Read(DRV_HANDLE handle, DRV_SDCARD_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SDCARD_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from the SD Card. The function returns with a valid buffer handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SDCARD_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the target buffer pointer is NULL
- if the number of blocks to be read is zero or more than the actual number of blocks available
- if a buffer object could not be allocated to the request
- if the client opened the driver in write only mode

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SDCARD_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SDCARD_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SDCARD driver instance.

DRV_SDCARD_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = 0x00;
uint32_t    nBlock = 2;
DRV_SDCARD_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySDCARDHandle is the handle returned
// by the DRV_SDCARD_Open function.

DRV_SDCARD_Read(mySDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read Successful
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

commandHandle	Pointer to an argument that will contain the return buffer handle
targetBuffer	Buffer into which the data read from the SD Card will be placed
blockStart	Start block address of the SD Card from where the read should begin.
nBlock	Total number of blocks to be read.

Function

void DRV_SDCARD_Read
(
const DRV_HANDLE handle,
DRV_SDCARD_COMMAND_HANDLE * commandHandle,
void * targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SDCARD_Write Function

Writes blocks of data starting at the specified address of the SD Card.

File

drv_sdcard.h

C

```
void DRV_SDCARD_Write(DRV_HANDLE handle, DRV_SDCARD_COMMAND_HANDLE * commandHandle, void * sourceBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SDCARD_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data to the SD Card. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SDCARD_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the source buffer pointer is NULL
- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SDCARD_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SDCARD_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = 0x00;
uint32_t    nBlock = 2;
DRV_SDCARD_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;
```

```
// mySDCARDHandle is the handle returned
// by the DRV_SDCARD_Open function.

// Client registers an event handler with driver

DRV_SDCARD_EventHandlerSet(mySDCARDHandle, APP_SDCARDEventHandler, (uintptr_t)&myAppObj);

DRV_SDCARD_Write(mySDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SDCARDEventHandler(DRV_SDCARD_EVENT event,
        DRV_SDCARD_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed to the SD Card.
blockStart	Start block address of SD Card where the writes should begin.
nBlock	Total number of blocks to be written.

Function

void DRV_SDCARD_Write
(
const DRV_HANDLE handle,
DRV_SDCARD_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SDCARD_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_sdcard.h

C

```
void DRV_SDCARD_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t
context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client queues a request for a read or a write operation, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read or write operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SDCARD_COMMAND_HANDLE commandHandle;

// drvSDCARDHandle is the handle returned
// by the DRV_SDCARD_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SDCARD_EventHandlerSet(drvSDCARDHandle, APP_SDCARDEventHandler, (uintptr_t)&myAppObj);

DRV_SDCARD_Read(drvSDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SDCARDEventHandler(DRV_SDCARD_EVENT event,
        DRV_SDCARD_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
```

```
            break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SDCARD_EventHandlerSet

(

const DRV_HANDLE handle,

const void * eventHandler,

const uintptr_t context

);

c) Status Functions

DRV_SDCARD_IsAttached Function

Returns the physical attach status of the SD Card.

File

drv_sdcard.h

C

```
bool DRV_SDCARD_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns the attach status of the SD Card. Returns true if the SD Card is attached and initialized by the SDCARD driver otherwise returns false.

Description

This function returns the physical attach status of the SD Card.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isSDCARDAttached;
isSDCARDAttached = DRV_SDCARD_isAttached(drvSDCARDHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SDCARD_IsAttached

(

const DRV_HANDLE handle

);

DRV_SDCARD_IsWriteProtected Function

Returns the write protect status of the SDCARD.

File

drv_sdcard.h

C

```
bool DRV_SDCARD_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Returns true if the attached SD Card is write protected. Returns false if the handle is not valid, or if the SD Card is not write protected.

Description

This function returns true if the SD Card is write protected otherwise it returns false.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isWriteProtected;
isWriteProtected = DRV_SDCARD_IsWriteProtected(drvSDCARDHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SDCARD_IsWriteProtected

(

const DRV_HANDLE handle

);

DRV_SDCARD_CommandStatus Function

Gets the current status of the command.

File

drv_sdcard.h

C

```
DRV_SDCARD_COMMAND_STATUS DRV_SDCARD_CommandStatus(const DRV_HANDLE handle, const DRV_SDCARD_COMMAND_HANDLE
commandHandle);
```

Returns

A DRV_SDCARD_COMMAND_STATUS value describing the current status of the command. Returns DRV_SDCARD_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to be polled on. The function may return DRV_SDCARD_COMMAND_HANDLE_INVALID in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another read or write request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive read or write operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called.

The DRV_SDCARD_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                      handle;         // Returned from DRV_SDCARD_Open
DRV_SDCARD_COMMAND_HANDLE       commandHandle;
DRV_SDCARD_COMMAND_STATUS       status;

status = DRV_SDCARD_CommandStatus(handle, commandHandle);
if(status == DRV_SDCARD_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_SDCARD_COMMAND_STATUS DRV_SDCARD_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SDCARD_COMMAND_HANDLE commandHandle

);

DRV_SDCARD_GeometryGet Function

Returns the geometry of the device.

File

drv_sdcard.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SDCARD_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SD Card.

- Media Property
- Number of Read/Write/Erase regions in the SD Card
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * SDCARDGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalSize;

SDCARDGeometry = DRV_SDCARD_GeometryGet(SDCARDOpenHandle1);
```

```
readBlockSize  = SDCARDGeometry->geometryTable->blockSize;
nReadBlocks = SDCARDGeometry->geometryTable->numBlocks;
nReadRegions = SDCARDGeometry->numReadRegions;

writeBlockSize  = (SDCARDGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (SDCARDGeometry->geometryTable +2)->blockSize;

totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_SDCARD_GeometryGet
(
const DRV_HANDLE handle
);

d) Data Types and Constants

DRV_SDCARD_INDEX_0 Macro

SD Card driver index definitions

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_0 0
```

Description

SD Card Driver Module Index Numbers

These constants provide SD Card driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SDCARD_Initialize and DRV_SDCARD_Open routines to identify the driver instance in use.

DRV_SDCARD_INDEX_COUNT Macro

Number of valid SD Card driver indices

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_COUNT DRV_SDCARD_INDEX_MAX
```

Description

SD Card Driver Module Index Count

This constant identifies number of valid SD Card driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_SDCARD_INIT Structure

Contains all the data necessary to initialize the SD Card device

File

drv_sdcard.h

C

```
typedef struct _DRV_SDCARD_INIT {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiIndex;
  SPI_MODULE_ID spiId;
  CLK_BUSES_PERIPHERAL spiClk;
  uint32_t sdcardSpeedHz;
  SDCARD_DETECTION_LOGIC sdCardPinActiveLogic;
  PORTS_CHANNEL cardDetectPort;
  PORTS_BIT_POS cardDetectBitPosition;
  PORTS_CHANNEL writeProtectPort;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPort;
  PORTS_BIT_POS chipSelectBitPosition;
} DRV_SDCARD_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiIndex;	SPI driver index
SPI_MODULE_ID spiId;	Identifies peripheral (PLIB-level) ID
CLK_BUSES_PERIPHERAL spiClk;	Peripheral clock used by the SPI
uint32_t sdcardSpeedHz;	SD card communication speed
SDCARD_DETECTION_LOGIC sdCardPinActiveLogic;	SD Card Pin Detection Logic
PORTS_CHANNEL cardDetectPort;	Card detect port
PORTS_BIT_POS cardDetectBitPosition;	Card detect pin
PORTS_CHANNEL writeProtectPort;	Write protect port
PORTS_BIT_POS writeProtectBitPosition;	Write protect pin
PORTS_CHANNEL chipSelectPort;	Chip select port
PORTS_BIT_POS chipSelectBitPosition;	Chip select pin

Description

SD Card Device Driver Initialization Data

This structure contains all the data necessary to initialize the SD Card device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SDCARD_Initialize routine.

SDCARD_DETECTION_LOGIC Enumeration

Defines the different system events

File

drv_sdcard.h

C

```
typedef enum {
  SDCARD_DETECTION_LOGIC_ACTIVE_LOW,
  SDCARD_DETECTION_LOGIC_ACTIVE_HIGH
} SDCARD_DETECTION_LOGIC;
```

Members

Members	Description
SDCARD_DETECTION_LOGIC_ACTIVE_LOW	The media event is SD Card attach

SDCARD_DETECTION_LOGIC_ACTIVE_HIGH	The media event is SD Card detach

Description

System events

This enum defines different system events.

Remarks

None.

SDCARD_MAX_LIMIT Macro

Maximum allowed SD card instances

File

drv_sdcard.h

C

```
#define SDCARD_MAX_LIMIT 2
```

Description

SD Card Driver Maximum allowed limit

This constant identifies number of valid SD Card driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_SDCARD_INDEX_1 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_1 1
```

Description

This is macro DRV_SDCARD_INDEX_1.

DRV_SDCARD_INDEX_2 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_2 2
```

Description

This is macro DRV_SDCARD_INDEX_2.

DRV_SDCARD_INDEX_3 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_3 3
```

Description

This is macro DRV_SDCARD_INDEX_3.

DRV_SDCARD_COMMAND_HANDLE_INVALID Macro

SDCARD Driver's Invalid Command Handle.

File

drv_sdcard.h

C

```
#define DRV_SDCARD_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SDCARD Driver Invalid Command Handle.

This value defines the SDCARD Driver Invalid Command Handle. This value is returned by read or write routines when the command request was not accepted.

Remarks

None.

DRV_SDCARD_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_sdcard.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SDCARD_COMMAND_HANDLE;
```

Description

SDCARD Driver command handle.

A command handle is returned by a call to the Read or Write functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SDCARD_COMMAND_STATUS Enumeration

Identifies the possible events that can result from a request.

File

drv_sdcard.h

C

```
typedef enum {
  DRV_SDCARD_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_SDCARD_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_SDCARD_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_SDCARD_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_SDCARD_COMMAND_STATUS;
```

Members

Members	Description
DRV_SDCARD_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_SDCARD_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started

DRV_SDCARD_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SDCARD_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

SDCARD Driver Events

This enumeration identifies the possible events that can result from a read or a write request made by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SDCARD_EventHandlerSet function when a request is completed.

DRV_SDCARD_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sdcard.h

C

```
typedef enum {
  DRV_SDCARD_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SDCARD_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_SDCARD_EVENT;
```

Members

Members	Description
DRV_SDCARD_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_SDCARD_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

SDCARD Driver Events

This enumeration identifies the possible events that can result from a read or a write request issued by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SDCARD_EventHandlerSet function when a request is completed.

DRV_SDCARD_EVENT_HANDLER Type

Pointer to a SDCARDDriver Event handler function

File

drv_sdcard.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_SDCARD_EVENT_HANDLER;
```

Returns

None.

Description

SDCARD Driver Event Handler Function Pointer

This data type defines the required function signature for the SDCARD event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SDCARD_EVENT_COMMAND_COMPLETE, it means that the write or a erase operation was completed successfully.

If the event is DRV_SDCARD_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_SDCARD_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

Example

```
void APP_MySDCARDEventHandler
(
    DRV_SDCARD_EVENT event,
    DRV_SDCARD_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write requests
context	Value identifying the context of the application that registered the event handling function

Files

Files

Name	Description
drv_sdcard.h	SD Card Device Driver Interface File
drv_sdcard_config_template.h	SD Card driver configuration definitions template

Description

This section lists the source and header files used by the SD Card Driver Library.

drv_sdcard.h

SD Card Device Driver Interface File

Enumerations

	Name	Description
	DRV_SDCARD_COMMAND_STATUS	Identifies the possible events that can result from a request.
	DRV_SDCARD_EVENT	Identifies the possible events that can result from a request.
	SDCARD_DETECTION_LOGIC	Defines the different system events

Functions

	Name	Description
	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_CommandStatus	Gets the current status of the command.
	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SDCARD_GeometryGet	Returns the geometry of the device.
	DRV_SDCARD_Initialize	Initializes the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_IsAttached	Returns the physical attach status of the SD Card.
	DRV_SDCARD_IsWriteProtected	Returns the write protect status of the SDCARD.
	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SDCARD_Read	Reads blocks of data from the specified block address of the SD Card.
	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_SDCARD_Status	Provides the current status of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic
	DRV_SDCARD_Write	Writes blocks of data starting at the specified address of the SD Card.

Macros

	Name	Description
	DRV_SDCARD_COMMAND_HANDLE_INVALID	SDCARD Driver's Invalid Command Handle.
	DRV_SDCARD_INDEX_0	SD Card driver index definitions
	DRV_SDCARD_INDEX_1	This is macro DRV_SDCARD_INDEX_1.
	DRV_SDCARD_INDEX_2	This is macro DRV_SDCARD_INDEX_2.
	DRV_SDCARD_INDEX_3	This is macro DRV_SDCARD_INDEX_3.
	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
	SDCARD_MAX_LIMIT	Maximum allowed SD card instances

Structures

	Name	Description
	_DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device

Types

	Name	Description
	DRV_SDCARD_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SDCARD_EVENT_HANDLER	Pointer to a SDCARDDriver Event handler function

Description

SD Card Device Driver Interface

The SD Card device driver provides a simple interface to manage the "SD Card" peripheral. This file defines the interface definitions and prototypes for the SD Card driver.

File Name

drv_sdcard.h

Company

Microchip Technology Inc.

drv_sdcard_config_template.h

SD Card driver configuration definitions template

Macros

	Name	Description
	DRV_SDCARD_CLIENTS_NUMBER	Selects the miximum number of clients
	DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK	Enable SD Card write protect check.
	DRV_SDCARD_INDEX_MAX	SD Card Static Index selection
	DRV_SDCARD_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver
	DRV_SDCARD_POWER_STATE	Defines an override of the power state of the SD Card driver.
	DRV_SDCARD_SYS_FS_REGISTER	Register to use with the File system

Description

SD Card Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

File Name

drv_sdcard_config_template.h

Company

Microchip Technology Inc.

SPI Driver Library

This section describes the Serial Peripheral Interface (SPI) Driver Library.

Introduction

This library provides an interface to manage the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers in different modes of operation.

Description

The SPI module is a full duplex synchronous serial interface useful for communicating with other peripherals or microcontrollers in master/slave relationship and it can transfer data over short distances at high speeds. The peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc. The SPI module is compatible with Motorola's SPI and SIOP interfaces.

During data transfer devices can work either in master or in Slave mode. The source of synchronization is the system clock, which is generated by the master. The SPI module allows one or more slave devices to be connected to a single master device via the same bus.

The SPI serial interface consists of four pins, which are further sub-divided into data and control lines:

Data Lines:

• MOSI – Master Data Output, Slave Data Input

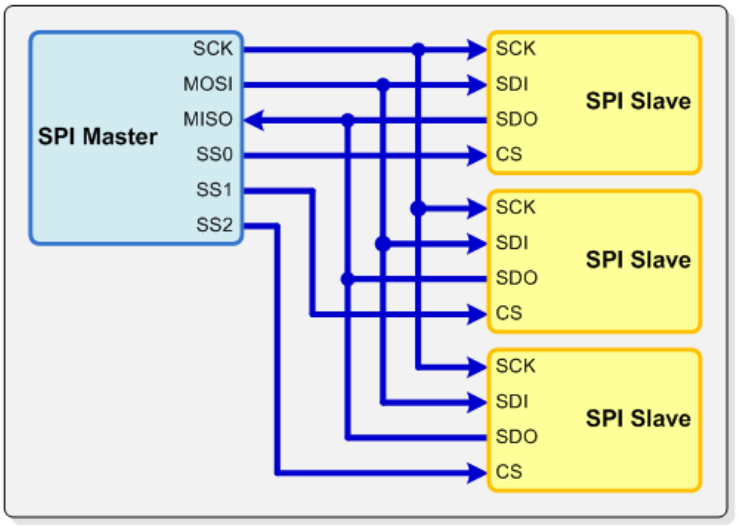
• MISO – Master Data Input, Slave Data Output

Control Lines:

• SCLK – Serial Clock

• /SS – Slave Select (no addressing)

SPI Master-Slave Relationship



The SPI module can be configured to operate using two, three, or four pins. In the 3-pin mode, the Slave Select line is not used. In the 2-pin mode, both the MOSI and /SS lines are not used.

Note: Third-party trademarks are property of their respective owners. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the SPI Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_spi.h`

The interface to the SPI Driver library is defined in the `drv_spi.h` header file. Any C language source (`.c`) file that uses the SPI Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the SPI Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

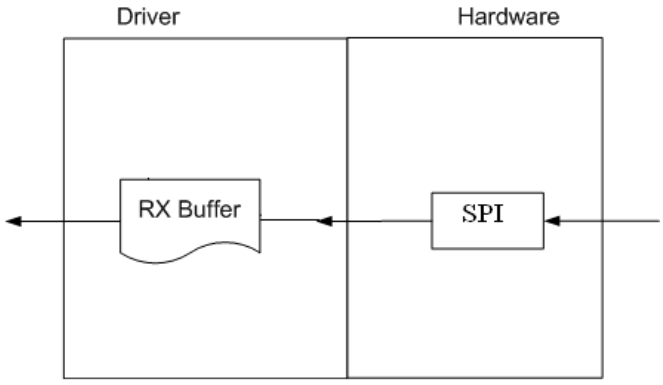
Description

Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not. The buffer depth varies across part families. The SPI driver abstracts out these differences and provides a unified model for data transfer across different types of SPIs available.

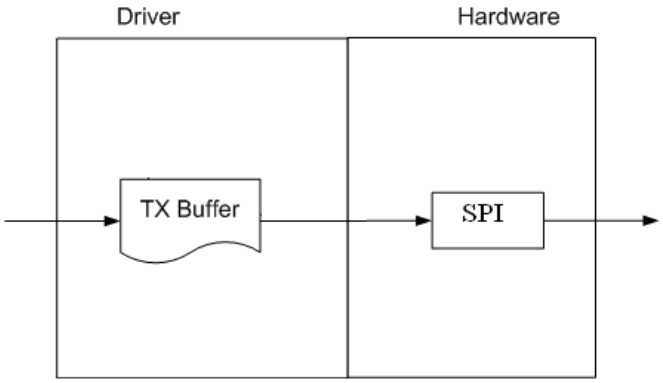
Both transmitter and receiver provides a buffer in the driver which transmits and receives data to/from the hardware. The SPI driver provides a set of interfaces to perform the read and the write.

The following diagrams illustrate the model used by the SPI driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Miscellaneous	Provides driver miscellaneous functions, data transfer status function, version identification functions etc.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the modes that are supported for your device.

System Access

System Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the SPI module would be initialized with the following configuration

settings (either passed dynamically at run time using DRV_SPI_INIT or by using Initialization Overrides) that are supported by the specific SPI device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., SPI_ID_2)
- Defining the respective interrupt sources for TX, RX, and Error Interrupt

The DRV_SPI_Initialize API returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_SPI_Deinitialize, DRV_SPI_Status, and DRV_SPI_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

Example:

```
DRV_SPI_INIT               spiInitData;
SYS_MODULE_OBJ             objectHandle;

    spiInitData.moduleInit.value    = SYS_MODULE_POWER_RUN_FULL;
    spiInitData.spiId               = SPI_ID_1;
    spiInitData.taskMode            = DRV_SPI_TASK_MODE_POLLED;
    spiInitData.spiMode             = DRV_SPI_MODE_MASTER;
    spiInitData.spiProtocolType     = DRV_SPI_PROTOCOL_TYPE_STANDARD;
    spiInitData.commWidth           = SPI_COMMUNICATION_WIDTH_8BITS;
    spiInitData.baudRate            = 5000;
    spiInitData.bufferType          = DRV_SPI_BUFFER_TYPE_STANDARD;
                                      // It is highly recommended to set this to
                                      // DRV_SPI_BUFFER_TYPE_ENHANCED for hardware
                                      // that supports it
    spiInitData.inputSamplePhase    = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
    spiInitData.clockMode           = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
    spiInitData.txInterruptSource   = INT_SOURCE_SPI_1_TRANSMIT;
    spiInitData.rxInterruptSource   = INT_SOURCE_SPI_1_RECEIVE;
    spiInitData.errInterruptSource  = INT_SOURCE_SPI_1_ERROR;
    spiInitData.queueSize = 10;
    spiInitData.jobQueueReserveSize = 1;

    objectHandle = DRV_SPI_Initialize(DRV_SPI_INDEX_1, (SYS_MODULE_INIT*)&spiInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
   {
    // Handle error
  }
```

Tasks Routine

The system will either call DRV_SPI_Tasks, from System Task Service (in a polled environment) or DRV_SPI_Tasks will be called from the ISR of the SPI.

Client Access

General Client Operation

For the application to start using an instance of the module, it must call the DRV_SPI_Open function. This provides the configuration required to open the SPI instance for operation. If the driver is deinitialized using the function DRV_SPI_Deinitialize, the application must call the DRV_SPI_Open function again to set up the instance of the SPI.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

After a client instance is opened, DRV_SPI_ClientConfigure can be called to set a client-specific bps, OperationStarting and OperationEnded callbacks. The OperationStarting callback will be called before the first bit is put onto the SPI bus, allowing for the slave select line to be toggled to active. The OperationEnded callback will be called after the last bit is received, allowing for the slave select line to be toggled to inactive. These two callbacks will be called from the ISR, if the SPI driver is operating in ISR mode, care should be taken that they do the minimum needed. For example, OSAL calls make cause exceptions in ISR context.

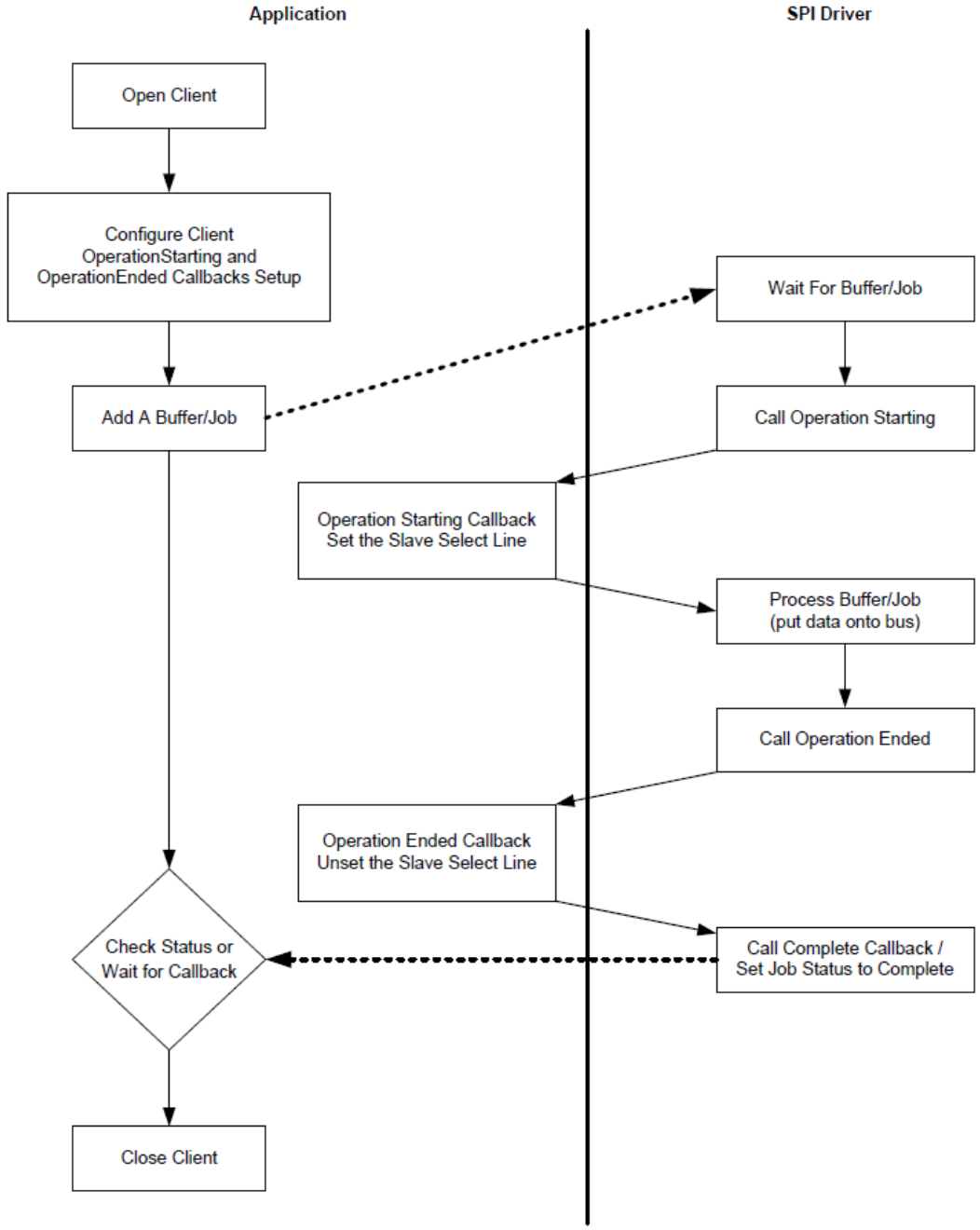
Example:

```
DRV_HANDLE handle;

// Configure the instance DRV_SPI_INDEX_1 with the configuration
handle = DRV_SPI_Open(DRV_SPI_INDEX_1, DRV_IO_INTENT_READWRITE);

if(handle == DRV_HANDLE_INVALID)
{
    // Client cannot open the instance.
}
```

Client Transfer - Core

Client basic functionality provides a extremely basic interface for the driver operation.

The following diagram illustrates the byte/word model used for the data transfer.



Note: It is not necessary to close and reopen the client between multiple transfers.

Client Data Transfer Functionality

Applications using the SPI byte/word functionality, need to perform the following:

1. The system should have completed necessary initialization and the DRV_SPI_Tasks should either be running in polled environment, or in an interrupt environment.
2. Open the driver using DRV_SPI_Open with the necessary intent.
3. Optionally configure the client with DRV_SPI_ClientConfigure to set up OperationStarting and OperationEnded callbacks to handle selecting and deselecting the slave select pin.
4. Add a buffer using the DRV_SPI_BufferAddRead/DRV_SPI_BufferAddWrite/DRV_SPI_BufferAddWriteRead functions. An optional callback can be provided that will be called when the buffer/job is complete.

5. Check for the current transfer status using DRV_SPI_BufferStatus until the transfer progress is DRV_SPI_BUFFER_EVENT_COMPLETE, or wait for the callback to be called. If the SPI driver is configured in Polled more, ensure that DRV_SPI_Tasks is called regularly to handle the buffer/job.
6. The client will be able to close the driver using DRV_SPI_Close when required.

Example:

```
SYS_MODULE_OBJ spiObject;

int main( void )
{
    while ( 1 )
    {
        appTask ();
        DRV_SPI_Tasks(spiObject);
    }
}
void appTask ()
{
    #define MY_BUFFER_SIZE   5
    DRV_HANDLE          handle;    // Returned from DRV_SPI_Open
    char          myBuffer[MY_BUFFER_SIZE] = { 11, 22, 33, 44, 55};
    unsigned int        numBytes;
    DRV_SPI_BUFFER_HANDLE bufHandle;

    // Preinitialize myBuffer with MY_BUFFER_SIZE bytes of valid data.
    while( 1 )
    {
        switch( state )
        {
            case APP_STATE_INIT:
                /* Initialize the SPI Driver */
                spiObject = DRV_SPI_Initialize( DRV_SPI_INDEX_1,
                                                ( SYS_MODULE_INIT * )
                                                  &initConf_1 );

                /* Check for the System Status */
                if( SYS_STATUS_READY != DRV_SPI_Status( spiObject ) )
                    return 0;

                /* Open the Driver */
                handle = DRV_SPI_Open( DRV_SPI_INDEX_1,
                                       DRV_IO_INTENT_EXCLUSIVE );

                /* Enable/Activate the CS */

                /* Update the state to transfer data */
                state = APP_STATE_DATA_PUT;
            break;

            case APP_STATE_DATA_PUT:
                bufHandle = DRV_SPI_BufferAddWrite ( handle, myBuffer,
                                                     5, NULL, NULL );
                /* Update the state to status check */
                state = APP_STATE_DATA_CHECK;
            break;

            case APP_STATE_DATA_CHECK:
                /* Check for the successful data transfer */
                if( DRV_SPI_BUFFER_EVENT_COMPLETE &
                    DRV_SPI_BufferStatus( bufhandle ) )
                {
                    /* Do this repeatedly */
                    state = APP_STATE_DATA_PUT;
                }

            break;
            default:
            break;
        }
    }
```

Configuring the Library

Miscellaneous Configuration

	Name	Description
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.

System Configuration

	Name	Description
	DRV_SPI_16BIT	Controls the compilation of 16 Bit mode
	DRV_SPI_32BIT	Controls the compilation of 32 Bit mode
	DRV_SPI_8BIT	Controls the compilation of 8 Bit mode
	DRV_SPI_DMA	Controls the compilation of DMA support
	DRV_SPI_DMA_DUMMY_BUFFER_SIZE	Controls the size of DMA dummy buffer
	DRV_SPI_DMA_TXFER_SIZE	Controls the size of DMA transfers
	DRV_SPI_EBM	Controls the compilation of Enhanced Buffer Mode mode
	DRV_SPI_ELEMENTS_PER_QUEUE	Controls the number of elements that are allocated.
	DRV_SPI_ISR	Controls the compilation of ISR mode
	DRV_SPI_MASTER	Controls the compilation of master mode
	DRV_SPI_POLLED	Controls the compilation of Polled mode
	DRV_SPI_RM	Controls the compilation of Standard Buffer mode
	DRV_SPI_SLAVE	Controls the compilation of slave mode

Description

The configuration of the SPI driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SPI driver. Based on the selections made, the SPI driver may support the selected features. These configuration settings will apply to all instances of the SPI driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

System Configuration

DRV_SPI_16BIT Macro

Controls the compilation of 16 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_16BIT 1
```

Description

SPI 16 Bit Mode Enable

This definition controls whether or not 16 Bit mode functionality is built as part of the driver. With it set to 1 then 16 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_16BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. With this set the BufferAdd functions will only accept buffer sizes of multiples of 2 (16 bit words)

Remarks

Optional definition

DRV_SPI_32BIT Macro

Controls the compilation of 32 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_32BIT 1
```

Description

SPI 32 Bit Mode Enable

This definition controls whether or not 32 Bit mode functionality is built as part of the driver. With it set to 1 then 32 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_32BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. With this set the BufferAdd functions will only accept buffer sizes of multiples of 4 (32 bit words)

Remarks

Optional definition

DRV_SPI_8BIT Macro

Controls the compilation of 8 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_8BIT 1
```

Description

SPI 8 Bit Mode Enable

This definition controls whether or not 8 Bit mode functionality is built as part of the driver. With it set to 1 then 8 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_8BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert.

Remarks

Optional definition

DRV_SPI_DMA Macro

Controls the compilation of DMA support

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA 1
```

Description

SPI DMA Enable

This definition controls whether or not DMA functionality is built as part of the driver. With it set to 1 then DMA will be compiled.

Remarks

Optional definition

DRV_SPI_DMA_DUMMY_BUFFER_SIZE Macro

Controls the size of DMA dummy buffer

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE 256
```

Description

SPI DMA Dummy Buffer Size

This controls the size of the buffer the SPI driver uses to give to the DMA service when it is to send and receive invalid data on the bus. This occurs when the number of bytes to be read are different than the number of bytes transmitted.

Remarks

Optional definition

DRV_SPI_DMA_TXFER_SIZE Macro

Controls the size of DMA transfers

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA_TXFER_SIZE 256
```

Description

SPI DMA Transfer Size

This definition controls the maximum number of bytes to transfer per DMA transfer.

Remarks

Optional definition

DRV_SPI_EBM Macro

Controls the compilation of Enhanced Buffer Mode mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_EBM 1
```

Description

SPI Enhanced Buffer Mode Enable (Hardware FIFO)

This definition controls whether or not Enhanced Buffer mode functionality is built as part of the driver. With it set to 1 then enhanced buffer mode will be compiled and bufferType = DRV_SPI_BUFFER_TYPE_ENHANCED will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. This mode is not available on all PIC32s. Trying to use this mode on PICMX3XX/4XX will cause compile time warnings and errors.

Remarks

Optional definition

DRV_SPI_ELEMENTS_PER_QUEUE Macro

Controls the number of elements that are allocated.

File

drv_spi_config_template.h

C

```
#define DRV_SPI_ELEMENTS_PER_QUEUE 10
```

Description

SPI Buffer Queue Depth

This definition along with DRV_SPI_INSTANCES_NUMBER and DRV_SPI_CLIENT_NUMBER controls how many buffer queue elements are created.

Remarks

Optional definition

DRV_SPI_ISR Macro

Controls the compilation of ISR mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_ISR 1
```

Description

SPI ISR Mode Enable

This definition controls whether or not ISR mode functionality is built as part of the driver. With it set to 1 then ISR mode will be compiled and taskMode = DRV_SPI_TASK_MODE_ISR will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_MASTER Macro

Controls the compilation of master mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_MASTER 1
```

Description

SPI Master Mode Enable

This definition controls whether or not master mode functionality is built as part of the driver. With it set to 1 then master mode will be compiled and spiMode = DRV_SPI_MODE_MASTER will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_POLLED Macro

Controls the compilation of Polled mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_POLLED 1
```

Description

SPI Polled Mode Enable

This definition controls whether or not polled mode functionality is built as part of the driver. With it set to 1 then polled mode will be compiled and taskMode = DRV_SPI_TASK_MODE_POLLED will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_RM Macro

Controls the compilation of Standard Buffer mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_RM 1
```

Description

SPI Standard Buffer Mode Enable

This definition controls whether or not Standard Buffer mode functionality is built as part of the driver. With it set to 1 then standard buffer mode will be compiled and bufferType = DRV_SPI_BUFFER_TYPE_STANDARD will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. This mode is available on all PIC32s

Remarks

Optional definition

DRV_SPI_SLAVE Macro

Controls the compilation of slave mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_SLAVE 1
```

Description

SPI Slave Mode Enable

This definition controls whether or not slave mode functionality is built as part of the driver. With it set to 1 then slave mode will be compiled and spiMode = DRV_SPI_MODE_SLAVE will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

Miscellaneous Configuration

DRV_SPI_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver .

File

drv_spi_config_template.h

C

```
#define DRV_SPI_INSTANCES_NUMBER 1
```

Description

SPI hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

Mandatory definition

DRV_SPI_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_spi_config_template.h

C

```
#define DRV_SPI_CLIENTS_NUMBER 1
```

Description

SPI maximum number of clients

This definition selects the maximum number of clients that the SPI driver can support at run time.

Remarks

Mandatory definition

Building the Library

This section lists the files that are available in the SPI Driver Library.

Description

This section list the files that are available in the \src folder of the SPI Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/spi`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_spi.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_spi.c	Basic SPI Driver implementation file.
/src/dynamic/drv_spi_api.c	Functions used by the driver API.
/src/drv_spi_sys_queue_fifo.c	Queue implementation used by the SPI Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SPI Driver Library depends on the following modules:

- Clock System Service Library

Optional Dependencies

- DMA System Service Library (used when operating in DMA mode)
- Interrupt System Service Library (used when task is running in Interrupt mode)

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_SPI_Initialize	Initializes the SPI instance for the specified driver index. **Implementation:** Static/Dynamic

	DRV_SPI_Deinitialize	Deinitializes the specified instance of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Status	Provides the current status of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Static/Dynamic

b) Client Setup Functions

	Name	Description
	DRV_SPI_Close	Closes an opened instance of the SPI driver. **Implementation:** Static/Dynamic
	DRV_SPI_Open	Opens the specified SPI driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_SPI_ClientConfigure	Configures a SPI client with specific data. **Implementation:** Static/Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_SPI_BufferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead2	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite2	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead2	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPIn_ReceiverBufferIsFull	Returns the receive buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static
	DRV_SPIn_TransmitterBufferIsFull	Returns the transmit buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static

Description

This section describes the API functions of the SPI Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_SPI_Initialize Function

Initializes the SPI instance for the specified driver index.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
SYS_MODULE_OBJ DRV_SPI_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- If successful - returns a valid handle to a driver instance object

- If unsuccessful - returns SYS_MODULE_OBJ_INVALID

Description

This routine initializes the SPI driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the SPI module ID. For example, driver instance 0 can be assigned to SPI2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_SPI_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other SPI routine is called.

This routine should only be called once during system initialization unless DRV_SPI_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_SPI_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the SPI initialization structure
init.spiId = SPI_ID_1,
init.taskMode = DRV_SPI_TASK_MODE_ISR,
init.spiMode = DRV_SPI_MODE_MASTER,
init.allowIdleRun = false,
init.spiProtocolType = DRV_SPI_PROTOCOL_TYPE_STANDARD,
init.commWidth = SPI_COMMUNICATION_WIDTH_8BITS,
init.baudClockSource = SPI_BAUD_RATE_PBCLK_CLOCK;
init.spiClk = CLK_BUS_PERIPHERAL_2,
init.baudRate = 10000000,
init.bufferType = DRV_SPI_BUFFER_TYPE_ENHANCED,
init.clockMode = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
init.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE,
init.txInterruptSource = INT_SOURCE_SPI_1_TRANSMIT,
init.rxInterruptSource = INT_SOURCE_SPI_1_RECEIVE,
init.errInterruptSource = INT_SOURCE_SPI_1_ERROR,
init.dummyByteValue = 0xFF,
init.queueSize = 10,
init.jobQueueReserveSize = 1,

objectHandle = DRV_SPI_Initialize(DRV_SPI_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the SPI id. The hardware SPI id is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_SPI_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_SPI_Deinitialize Function

Deinitializes the specified instance of the SPI driver module.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_SPI_Status operation. The system has to use DRV_SPI_Status to find out when the module is in the ready state.

Preconditions

Function DRV_SPI_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_SPI_Initialize
SYS_STATUS          status;

DRV_SPI_Deinitialize ( object );

status = DRV_SPI_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Function

void DRV_SPI_Deinitialize (SYS_MODULE_OBJ object)

DRV_SPI_Status Function

Provides the current status of the SPI driver module.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
SYS_STATUS DRV_SPI_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is busy with a previous

system level operation and cannot start another

Description

This function provides the current status of the SPI driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_SPI_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SPI_Initialize
SYS_STATUS          status;

status = DRV_SPI_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Function

SYS_STATUS DRV_SPI_Status (SYS_MODULE_OBJ object)

DRV_SPI_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks() function. In interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SPI_Initialize

while( true )
{
    DRV_SPI_Tasks ( object );

    // Do other tasks
```

```
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SPI_Initialize)

Function

void DRV_SPI_Tasks (SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_SPI_Close Function

Closes an opened instance of the SPI driver.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of the SPI driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SPI_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SPI_Open

DRV_SPI_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SPI_Close (DRV_HANDLE handle)

DRV_SPI_Open Function

Opens the specified SPI driver instance and returns a handle to it.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_HANDLE DRV_SPI_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_SPI_INSTANCES_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified SPI driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_SPI_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_SPI_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_SPI_Open( DRV_SPI_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_SPI_Initialize(). Please note this is not the SPI ID.
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SPI_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent)

DRV_SPI_ClientConfigure Function

Configures a SPI client with specific data.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
int32_t DRV_SPI_ClientConfigure(DRV_HANDLE handle, const DRV_SPI_CLIENT_DATA * cfgData);
```

Returns

- If successful - the routing will return greater than or equal to zero
- If an error occurs - the return value is negative

Description

This routine takes a DRV_SPI_CLIENT_DATA structure and sets client specific options. Whenever a new SPI job is started these values will be used. Passing in NULL will reset the client back to configuration parameters passed to driver initialization. A zero in any of the structure elements

will reset that specific configuration back to the driver default.

Preconditions

The DRV_SPI_Open function must have been called before calling this function.

Parameters

Parameters	Description
handle	handle of the client returned by DRV_SPI_Open.
cfgData	Client-specific configuration data.

Function

int32_t DRV_SPI_ClientConfigure (DRV_HANDLE handle,
const DRV_SPI_CLIENT_DATA * cfgData)

c) Data Transfer Functions

DRV_SPI_BufferStatus Function

Returns the transmitter and receiver transfer status.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_EVENT DRV_SPI_BufferStatus(DRV_SPI_BUFFER_HANDLE bufferHandle);
```

Returns

A DRV_SPI_BUFFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_SPI_BUFFER_STATUS enumeration set. The caller should perform an AND with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_SPI_BufferAddmust have been called to obtain the buffer handle associated with that transfer.

Example

```
// Buffer handle returned from the data transfer function
DRV_SPI_BUFFER_HANDLE bufferHandle;

if(DRV_SPI_BufferStatus(bufferHandle) == DRV_SPI_BUFFER_EVENT_COMPLETE)
{
    // All transmitter data has been sent.
}
```

Parameters

Parameters	Description
bufferHandle	A valid buffer handle, returned from the driver's data transfer routine

Function

DRV_SPI_BUFFER_EVENT DRV_SPI_BufferStatus (DRV_SPI_BUFFER_HANDLE bufferHandle)

DRV_SPI_BufferAddRead Function

Registers a buffer for a read operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead(DRV_HANDLE handle, void * rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddRead2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddRead( handle, myBuffer, 10, NULL, NULL );
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read from the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete.
context	unused by the driver but this is passed to the callback when it is called.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead (DRV_HANDLE handle, void *rxBuffer,

size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,

void * context)

DRV_SPI_BufferAddWrite Function

Registers a buffer for a write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite(DRV_HANDLE handle, void * txBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddWrite2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWrite( handle, myBuffer, 10, NULL, NULL );
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
size	Number of bytes to be written to the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite (DRV_HANDLE handle, void *txBuffer,

size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context)

DRV_SPI_BufferAddWriteRead Function

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead(DRV_HANDLE handle, void * txBuffer, size_t txSize, void *
rxBuffer, size_t rxSize, DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddWriteRead2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myReadBuffer[MY_BUFFER_SIZE], myWriteBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWriteRead( handle, myWriteBuffer, 10, myReadBuffer, 10, NULL, NULL
);
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
txSize	Number of bytes to be written to the SPI bus.
rxBuffer	The buffer to which the data should be written to.
rxSize	Number of bytes to be read from the SPI bus
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead(DRV_HANDLE handle,
void *txBuffer, void *rxBuffer, size_t size,)

DRV_SPI_BufferAddRead2 Function

Registers a buffer for a read operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead2(DRV_HANDLE handle, void * rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE * jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddRead2( handle, myBuffer, 10, NULL, NULL, &bufferHandle2 );
        if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle2) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read from the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called

jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead2 (DRV_HANDLE handle, void *rxBuffer,

size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,

void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPI_BufferAddWrite2 Function

Registers a buffer for a write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite2(DRV_HANDLE handle, void * txBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE * jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWrite2( handle, myBuffer, 10, NULL, NULL, &bufferHandle2 );
        if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle2) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.

size	Number of bytes to be written to the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called
jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite2 (DRV_HANDLE handle, void *txBuffer,

size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,

void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPI_BufferAddWriteRead2 Function

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead2(DRV_HANDLE handle, void * txBuffer, size_t txSize, void *
rxBuffer, size_t rxSize, DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE *
jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myReadBuffer[MY_BUFFER_SIZE], myWriteBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWriteRead2( handle, myWriteBuffer, 10, myReadBuffer, 10, NULL,
NULL, &bufferHandle2 );
        if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if(DRV_SPI_BufferStatus(bufferHandle2) == DRV_SPI_BUFFER_EVENT_COMPLETE)
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
txSize	Number of bytes to be written to the SPI bus.
rxBuffer	The buffer to which the data should be written to.
rxSize	Number of bytes to be read from the SPI bus
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called
jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead2(DRV_HANDLE handle,
void *txBuffer, void *rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPIn_ReceiverBufferIsFull Function

Returns the receive buffer status. 'n' represents the instance of the SPI driver used.

Implementation: Static

File

drv_spi.h

C

```
bool DRV_SPIn_ReceiverBufferIsFull();
```

Returns

Receive Buffer Status

- 1 - Full
- 0 - Empty

Description

This function returns the receive buffer status (full/empty).

Remarks

None.

Preconditions

None.

Example

```
bool rxBufStat;
// Using instance 1 of SPI driver, that is n = 1
rxBufStat = DRV_SPI1_ReceiverBufferIsFull();

if (rxBufStat)
{
...
}
```

Function

bool DRV_SPIn_ReceiverBufferIsFull(void)

DRV_SPIn_TransmitterBufferIsFull Function

Returns the transmit buffer status. 'n' represents the instance of the SPI driver used.

Implementation: Static

File

drv_spi.h

C

```
bool DRV_SPIn_TransmitterBufferIsFull();
```

Returns

Transmit Buffer Status

- 1 - Full
- 0 - Empty

Description

This function returns the transmit buffer status (full/empty).

Remarks

None.

Preconditions

None.

Example

```
bool txBufStat;
// Using instance 1 of SPI driver, that is n = 1
txBufStat = DRV_SPI1_TransmitterBufferIsFull();

if (txBufStat)
{
...
}
```

Function

bool DRV_SPIn_TransmitterBufferIsFull(void)

d) Miscellaneous Functions

e) Data Types and Constants

Files

Files

Name	Description
drv_spi.h	SPI device driver interface file.
drv_spi_config_template.h	SPI Driver configuration definitions template.

Description

This section lists the source and header files used by the SPI Driver Library.

drv_spi.h

SPI device driver interface file.

Functions

	Name	Description
	DRV_SPI_BufferAddRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead2	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic

	DRV_SPI_BufferAddWrite	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite2	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead2	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_SPI_ClientConfigure	Configures a SPI client with specific data. **Implementation:** Static/Dynamic
	DRV_SPI_Close	Closes an opened instance of the SPI driver. **Implementation:** Static/Dynamic
	DRV_SPI_Deinitialize	Deinitializes the specified instance of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Initialize	Initializes the SPI instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_SPI_Open	Opens the specified SPI driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_SPI_Status	Provides the current status of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_SPIn_ReceiverBufferIsFull	Returns the receive buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static
	DRV_SPIn_TransmitterBufferIsFull	Returns the transmit buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static

Description

SPI Driver Interface

The SPI driver provides a simple interface to manage the SPI module. This file defines the interface definitions and prototypes for the SPI driver.

File Name

drv_spi.h

Company

Microchip Technology Inc.

drv_spi_config_template.h

SPI Driver configuration definitions template.

Macros

	Name	Description
	DRV_SPI_16BIT	Controls the compilation of 16 Bit mode
	DRV_SPI_32BIT	Controls the compilation of 32 Bit mode
	DRV_SPI_8BIT	Controls the compilation of 8 Bit mode
	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_SPI_DMA	Controls the compilation of DMA support
	DRV_SPI_DMA_DUMMY_BUFFER_SIZE	Controls the size of DMA dummy buffer
	DRV_SPI_DMA_TXFER_SIZE	Controls the size of DMA transfers
	DRV_SPI_EBM	Controls the compilation of Enhanced Buffer Mode mode
	DRV_SPI_ELEMENTS_PER_QUEUE	Controls the number of elements that are allocated.
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_ISR	Controls the compilation of ISR mode

	DRV_SPI_MASTER	Controls the compilation of master mode
	DRV_SPI_POLLED	Controls the compilation of Polled mode
	DRV_SPI_RM	Controls the compilation of Standard Buffer mode
	DRV_SPI_SLAVE	Controls the compilation of slave mode

Description

SPI Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_spi_config_template.h

Company

Microchip Technology Inc.

SPI Flash Driver Library

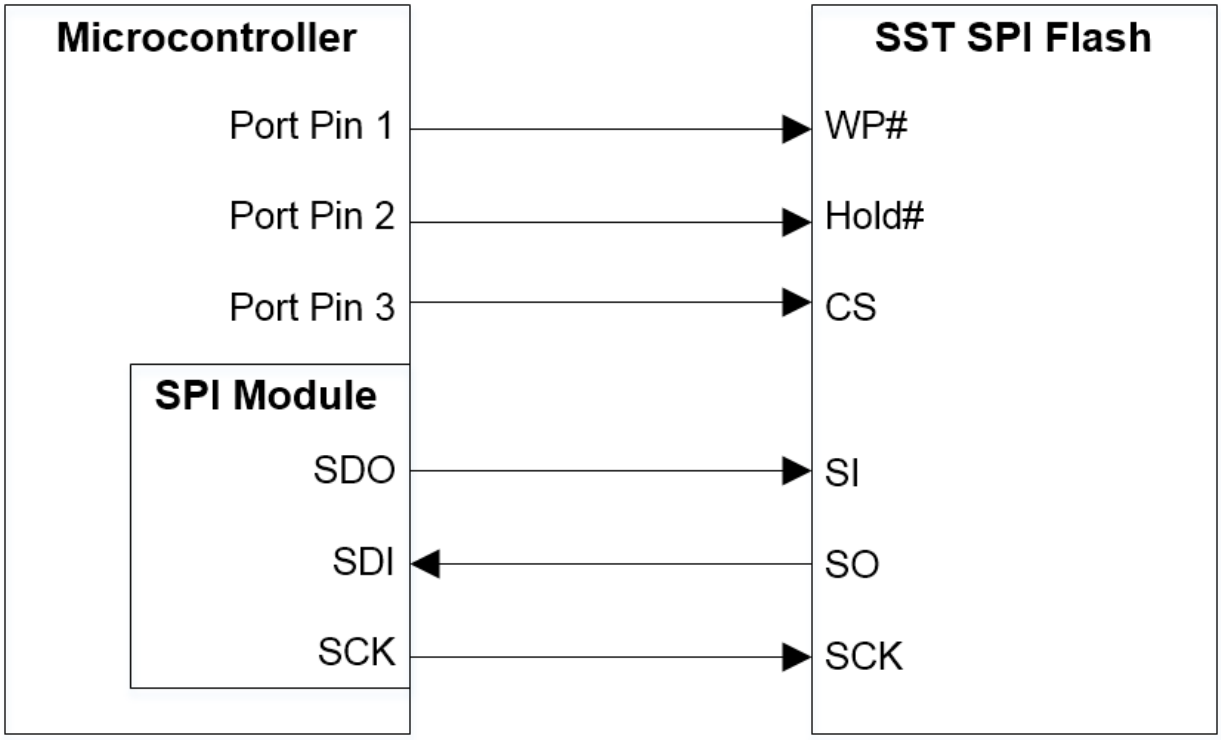
This section describes the Serial Peripheral Interface (SPI) Flash Driver Library.

Introduction

This library provides an interface to manage the SST SPI Flash modules (SST25VF020B, SST25VF016B, and SST25VF064C) in different modes of operation.

Description

The SPI Flash Driver uses SPI interface to establish the communication between SST Flash and Microchip microcontrollers. The SPI module of the controller works as a Master device and the Flash module works as a Slave. The following diagram shows the pin connections that are required to make the driver operational:



The SPI Flash Driver is dynamic in nature, so single instance of it can support multiple clients that want to use the same Flash. Multiple instances of the driver can be used when multiple Flash devices are required to be part of the system. The SPI Driver, which is used by the SPI Flash Driver, can be configured for use in either Polled or Interrupt mode.

Using the Library

This topic describes the basic architecture of the SPI Flash Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_sst25vf016b.h`, `drv_sst25vf020b.h`, or `drv_sst25vf064c.h`

The interface to the SPI Flash Driver Library is defined in the header file. Any C language source (`.c`) file that uses the SPI Flash Driver library should include this header.

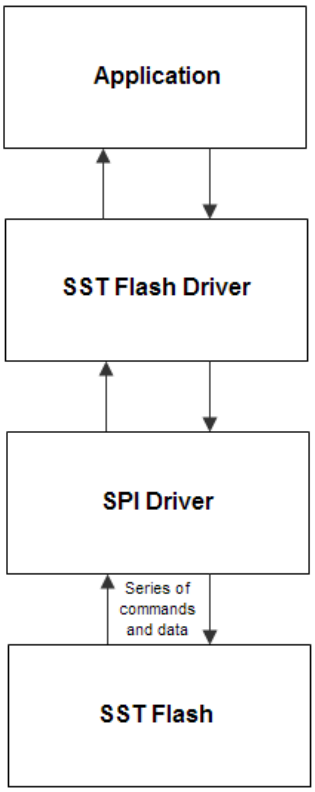
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the SPI Flash Driver Library with a convenient C language interface. This topic describes how that abstraction is modeled in software.

Description

The SST SPI Flash needs a specific set of commands to be given on its SPI interface along with the required address and data to do different operations. This driver abstracts these requirements and provide simple APIs that can be used to perform Erase, Write, and Read operations. The SPI Driver is used for this purpose. The following layered diagram depicts the communication between different modules.



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI Flash module.

Library Interface Section	Description
System Functions	These functions are accessed by the MPLAB Harmony System module and allow the driver to be initialized, deinitialized, and maintained.
Core Client Functions	These functions allow the application client to open and close the driver.
Block Operation Functions	These functions enable the Flash module to be erased, written, and read (to/from).
Media Interface Functions	These functions provide media status and the Flash geometry.

How the Library Works

The library provides interfaces to support:

- System Initialization/Deinitialization
- Opening the Driver
- Block Operations

System Initialization and Deinitialization

Provides information on initializing the system.

Description

System Initialization and Deinitialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization each instance of the SST Flash module would be initialized with the following configuration settings (either passed dynamically at run-time using DRV_SST25VF020B_INIT, DRV_SST25VF016B_INIT, or DRV_SST25VF064C_INIT, or by using Initialization Overrides) that are supported or used by the specific SST Flash device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section
- The SPI Driver Module Index which is intended to be used to communicate with SST Flash (e.g., DRV_SPI_INDEX_0)
- Port Pins of the microcontroller to be used for Chip Select, Write Protection, and Hold operations on the SST Flash device
- Maximum Buffer Queue Size for that instance of the SST Flash Driver

Using the SST25VF020B as an example, the DRV_SST25VF020B_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_SST25VF020B_Deinitialize, DRV_SST25VF020B_Status, and DRV_SST25VF020B_Tasks.

Note: The system initialization and the deinitialization settings, only affect the instance of the peripheral that is being initialized or deinitialized.

Example:

```
// This code example shows the initialization of the SST25VF020B SPI Flash
// Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2,
// and 3 of PORTB are configured for the Hold pin, Write Protection pin, and
// the Chip Select pin, respectively. The maximum buffer queue size is set to 5.

   DRV_SST25VF020B_INIT   SST25VF020BInitData;
   SYS_MODULE_OBJ      objectHandle;

   SST25VF020BInitData.moduleInit.value        = SYS_MODULE_POWER_RUN_FULL;
   SST25VF020BInitData.spiDriverModuleIndex    = DRV_SPI_INDEX_0;
   SST25VF020BInitData.holdPortChannel         = PORT_CHANNEL_B;
   SST25VF020BInitData.holdBitPosition         = PORTS_BIT_POS_1;
   SST25VF020BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
   SST25VF020BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
   SST25VF020BInitData.chipSelectPortChannel   = PORT_CHANNEL_F;
   SST25VF020BInitData.chipSelectBitPosition   = PORTS_BIT_POS_2;
   SST25VF020BInitData.queueSize               = 5;

   objectHandle = DRV_SST25VF020B_Initialize(DRV_SST25VF020B_INDEX_0,
                                  (SYS_MODULE_INIT*)SST25VF020BInitData);
   if (SYS_MODULE_OBJ_INVALID == objectHandle)
   {
       // Handle error
   }
```

Tasks Routine

The system will either call DRV_SST25VF020B_Tasks, from SYS_Tasks (in a polled environment) or DRV_SST25VF020B_Tasks will be called from the ISR of the SPI module in use.

Opening the Driver

Provides information on opening the driver.

Description

To use the SST Flash driver, the application must open the driver. Using the SST25VF020B as an example, this is done by calling the DRV_SST25VF020B_Open function. Calling this function with DRV_IO_INTENT_NONBLOCKING will cause the driver to be opened in non blocking mode. Then DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite and DRV_SST25VF020B_BlockRead functions when called by this client will be non-blocking.

The client can also open the driver in Read-only mode (DRV_IO_INTENT_READ), Write-only mode (DRV_IO_INTENT_WRITE), and Exclusive mode (DRV_IO_INTENT_EXCLUSIVE). If the driver has been opened exclusively by a client, it cannot be opened again by another client.

If successful, the DRV_SST25VF020B_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_SST25VF020B_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well.

The following code shows an example of the driver being opened in different modes.

```
DRV_HANDLE sstHandle1, sstHandle2;

/* Client 1 opens the SST driver in non blocking mode */
sstHandle1 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0, DRV_IO_INTENT_NONBLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == sstHandle1)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* Client 2 opens the SST driver in Exclusive Write only mode */
sstHandle2 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0, DRV_IO_INTENT_WRITE | DRV_IO_INTENT_EXCLUSIVE);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == sstHandle2)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}
```

Block Operations

Provides information on block operations.

Description

This driver provides simple client interfaces to Erase, Write, and Read the SST flash in blocks. A block is the unit to represent minimum amount of data that can be erased, written, or read. Block size may differ for Erase, Write, and Read operations. Using the SST25VF020B as an example, the DRV_SST25VF020B_GeometryGet function can be used to determine the different block sizes for the driver.

The DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite, and DRV_SST25VF020B_BlockRead functions are used to erase, write, and read the data to/from SST SPI Flash. These functions are always non-blocking. All of these functions follow a standard queue model to read, write, and erase. When any of these functions are called (i.e., a block request is made), the request is queued. The size of the queue is determined by the queueSize member of the DRV_SST25VF020B_INIT data structure. All of the requests in the queue are executed by the DRV_SST25VF020B_Tasks function one-by-one.

When the driver adds a request to the queue, it returns a buffer handle. This handle allows the client to track the request as it progresses through the queue. The buffer handle expires when the event associated with the buffer completes. The driver provides driver events (DRV_SST25VF020B_BLOCK_EVENT) that indicate termination of the buffer requests.

The following steps can be performed for a simple Block Data Operation:

1. The system should have completed necessary initialization of the SPI Driver and the SST Flash Driver, and the DRV_SST25VF020B_Tasks function should be running in a polled environment.
2. The DRV_SPI_Tasks function should be running in either a polled environment or an interrupt environment.
3. Open the driver using DRV_SST25VF020B_Open with the necessary intent.
4. Set an event handler callback using the function DRV_SST25VF020B_BlockEventHandlerSet.
5. Request for block operations using the functions, DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite, and DRV_SST25VF020B_BlockRead, with the appropriate parameters.
6. Wait for event handler callback to occur and check the status of the block operation using the callback function parameter of type DRV_SST25VF020B_BLOCK_EVENT.
7. The client will be able to close the driver using the function, DRV_SST25VF020B_Close, when required.

Example:

```
/* This code example shows usage of the block operations
```

```
 * on the SPI Flash SST25VF020B device */

DRV_HANDLE sstHandle1;
uint8_t myData1[10], myData2[10];
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE  blockHandle1, blockHandle2, blockHandle3;

/* The driver is opened for read-write in Exclusive mode */
sstHandle1 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0,
                                  DRV_IO_INTENT_READWRITE | DRV_IO_INTENT_EXCLUSIVE);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == sstHandle1)
{
    /* The driver could not be opened successfully */
}

/* Register a Buffer Event Handler with SST25VF020B  driver.
 * This event handler function will be called whenever
 * there is a buffer event.  An application defined
 * context can also be specified. This is returned when
 * the event handler is called.
 * */
DRV_SST25VF020B_BlockEventHandlerSet(sstHandle1,
        APP_SSTBufferEventHandler, NULL);

/* Request for all the three block operations one by one */

/* first block API to erase 1 block of the flash starting from address 0x0, each block is of 4kbyte */
DRV_SST25VF020B_BlockErase(sstHandle1, &blockHandle1, 0x0, 1);
/* 2nd block API to write myData1 in the first 10 locations of the flash */
DRV_SST25VF020B_BlockWrite(sstHandle1, &blockHandle2, &myData1[0], 0x0, 10);
/* 3rd block API to read the first 10 locations of the flash into myData2 */
DRV_SST25VF020B_BlockRead(sstHandle1, &blockHandle3, &myData2[0], 0x0, 10);

/* This is the Driver Event Handler */

void APP_SSTBufferEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE blockHandle, uintptr_t contextHandle)
{
    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:
    if ( blockHandle == blockHandle3)
    {
            /* This means the data was read */
    /* Do data verification/processing */
    }
            break;
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:
            /* Error handling here. */
            break;
        default:
            break;
    }
}
```

Configuring the Library

SST25VF016B Configuration

	Name	Description
	DRV_SST25VF016B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF016B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.

	DRV_SST25VF016B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_SST25VF016B_MODE	Determines whether the driver is implemented as static or dynamic
	DRV_SST25VF016B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

SST25VF020B Configuration

	Name	Description
	DRV_SST25VF020B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF020B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF020B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF020B_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF020B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

SST25VF064C Configuration

	Name	Description
	DRV_SST25VF064C_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF064C_HARDWARE_HOLD_ENABLE	Specifies whether or not the hardware hold feature is enabled.
	DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies whether or not the hardware write protect feature is enabled.
	DRV_SST25VF064C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF064C_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF064C_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

The SST Flash Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SST25VF016B Configuration

DRV_SST25VF016B_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_CLIENTS_NUMBER 4
```

Description

SST25VF016B Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF016B-1 will be accessed by 2 clients and SST25VF016B-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF016B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF016B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF016B_HARDWARE_HOLD_ENABLE Macro

Specifies if the hardware hold feature is enabled or not.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF016B Hardware HOLD Support

This macro defines if the hardware hold feature is enabled or not. If hardware hold is enabled, then user must provide a port pin corresponding to HOLD pin on the flash

Remarks

None

DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies if the hardware write protect feature is enabled or not.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF016B Hardware Write Protect Support

This macro defines if the hardware Write Protect feature is enabled or not. If hardware write protection is enabled, then user must provide a port pin corresponding to WP pin on the flash

Remarks

None.

DRV_SST25VF016B_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_INSTANCES_NUMBER 2
```

Description

SST25VF016B driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF016B modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF016B_MODE Macro

Determines whether the driver is implemented as static or dynamic

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_MODE DYNAMIC
```

Description

SST25VF016B mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF016B_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF016B Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF016B driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

SST25VF020B Configuration

DRV_SST25VF020B_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_CLIENTS_NUMBER 4
```

Description

SST25VF020B Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF020B-1 will be accessed by 2 clients and SST25VF020B-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF020B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF020B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF020B_HARDWARE_HOLD_ENABLE Macro

Specifies if the hardware hold feature is enabled or not.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF020B Hardware HOLD Support

This macro defines if the hardware hold feature is enabled or not. If hardware hold is enabled, then user must provide a port pin corresponding to HOLD pin on the flash

Remarks

None.

DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies if the hardware write protect feature is enabled or not.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF020B Hardware Write Protect Support

This macro defines if the hardware Write Protect feature is enabled or not. If hardware write protection is enabled, then user must provide a port pin corresponding to WP pin on the flash

Remarks

None.

DRV_SST25VF020B_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_INSTANCES_NUMBER 2
```

Description

SST25VF020B driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF020B modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF020B_MODE Macro

Determines whether the driver is implemented as static or dynamic.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_MODE DYNAMIC
```

Description

SST25VF020B mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF020B_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF020B Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF020B driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

SST25VF064C Configuration

DRV_SST25VF064C_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_CLIENTS_NUMBER 4
```

Description

SST25VF064C Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF064C-1 will be accessed by 2 clients and SST25VF064C-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF064C_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF064C_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF064C_HARDWARE_HOLD_ENABLE Macro

Specifies whether or not the hardware hold feature is enabled.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF064C Hardware HOLD Support

This macro defines whether or not the hardware hold feature is enabled. If hardware hold is enabled, the user must provide a port pin corresponding to the HOLD pin on the Flash device.

Remarks

None.

DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies whether or not the hardware write protect feature is enabled.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF064C Hardware Write Protect Support

This macro defines whether or not the hardware Write Protect feature is enabled. If hardware write protection is enabled, the user must provide a port pin corresponding to the WP pin on the Flash device.

Remarks

None.

DRV_SST25VF064C_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_INSTANCES_NUMBER 2
```

Description

SST25VF064C driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF064C modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF064C_MODE Macro

Determines whether the driver is implemented as static or dynamic.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_MODE DYNAMIC
```

Description

SST25VF064C mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF064C_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF064C Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF064C driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

Building the Library

This section lists the files that are available in the SPI Flash Driver Library.

Description

This section list the files that are available in the `/src` folder of the SPI Flash Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/spi_flash`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sst25vf016b/drv_sst25vf016b.h`	Header file that exports the SST25VF016B driver API.
`sst25vf020b/drv_sst25vf020b.h`	Header file that exports the SST25VF020B driver API.
`sst25vf064c/drv_sst25vf064c.h`	Header file that exports the SST25VF064C driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`sst25vf016b/src/dynamic/drv_sst25vf016b.c`	Basic SPI Flash Driver SST25VF016B implementation file.

sst25vf020b/src/dynamic/drv_sst25vf020b.c	Basic SPI Flash Driver SST25VF020B implementation file.
sst25vf064c/src/dynamic/drv_sst25vf064c.c	Basic SPI Flash Driver SST25VF064C implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
sst25vf020b/src/dynamic/drv_sst25vf020b_erasewrite.c	This file implements an optional BlockEraseWrite feature for the SST25VF020B driver.

Module Dependencies

The SPI Flash Driver Library depends on the following modules:

- SPI Driver Library
- Ports System Service Library

Library Interface

This section describes the API functions of the SPI Flash Driver Library.

Refer to each section for a detailed description.

SST25FV016B API

a) System Functions

	Name	Description
	DRV_SST25VF016B_Initialize	Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF016B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

b) Core Client Functions

	Name	Description
	DRV_SST25VF016B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF016B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic

c) Block Operation Functions

	Name	Description
	DRV_SST25VF016B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic

d) Media Interface Functions

	Name	Description
	DRV_SST25VF016B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF016B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF016B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF016B_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_SST25VF016B_EVENT_HANDLER	Pointer to a SST25VF016B SPI Flash Driver Event handler function. **Implementation:** Dynamic
	DRV_SST25VF016B_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF016B_INDEX_0	SPI Flash driver index definitions
	DRV_SST25VF016B_INDEX_1	This is macro DRV_SST25VF016B_INDEX_1.

Description

This section contains the SST25V016B Flash device API.

a) System Functions

DRV_SST25VF016B_Initialize Function

Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_MODULE_OBJ DRV_SST25VF016B_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF016B_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF016B SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
```

```
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF016B_INIT   SST25VF016BInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF016BInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF016BInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF016BInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF016BInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF016BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF016BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF016BInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF016BInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF016BInitData.queueSize = 5;

objectHandle = DRV_SST25VF016B_Initialize(DRV_SST25VF016B_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF016BInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SST25VF016B_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_SST25VF016B_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF016B_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF016B_Initialize
SYS_STATUS          status;
```

```
DRV_SST25VF016B_Deinitialize(object);

status = DRV_SST25VF016B_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF016B_Initialize

Function

void DRV_SST25VF016B_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF016B_Status Function

Gets the current status of the SPI Flash Driver module.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_STATUS DRV_SST25VF016B_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF016B_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF016B_Initialize
SYS_STATUS          SST25VF016BStatus;

SST25VF016BStatus = DRV_SST25VF016B_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF016BStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF016B_Initialize

Function

SYS_STATUS DRV_SST25VF016B_Status(SYS_MODULE_OBJ object)

DRV_SST25VF016B_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF016B_Initialize

while (true)
{
    DRV_SST25VF016B_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF016B_Initialize)

Function

void DRV_SST25VF016B_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF016B_Close Function

Closes an opened-instance of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF016B_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF016B_Open

DRV_SST25VF016B_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF016B_Close(DRV_Handle handle);

DRV_SST25VF016B_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
DRV_HANDLE DRV_SST25VF016B_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF016B_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

The driver status becomes ready inside "DRV_SST25VF016B_Tasks" function. To make the SST Driver status ready and hence successfully "Open" the driver, "Task" routine need to be called periodically.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF016B_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF016B_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF016B_Open(DRV_SST25VF016B_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
```

```
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST25VF016B_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
);

DRV_SST25VF016B_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
DRV_SST25VF016B_CLIENT_STATUS DRV_SST25VF016B_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF016B_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF016B_Open
DRV_SST25VF016B_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF016B_ClientStatus(handle);
if(DRV_SST25VF016B_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_SST25VF016B_CLIENT_STATUS DRV_SST25VF016B_ClientStatus(DRV_HANDLE handle);

c) Block Operation Functions

DRV_SST25VF016B_BlockErase Function

Erase the specified number of blocks in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockErase(const DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF016B_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF016B_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockErase( mySST25VF016BHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
```

```
        // contextHandle points to myAppObj.

        switch(event)
        {
            case DRV_SST25VF016B_EVENT_ERASE_COMPLETE:

                // This means the data was transferred.
                break;

            case DRV_SST25VF016B_EVENT_ERASE_ERROR:

                // Error handling here.

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF016B memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_SST25VF016B_BlockErase

(

const DRV_HANDLE handle,

DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,

uint32_t nBlock

);

DRV_SST25VF016B_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF016B_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF016B_BlockEventHandlerSet( mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF016B_BlockRead( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF016B_BlockEventHandlerSet

```
(
const      DRV_HANDLE handle,
const      DRV_SST25VF016B_EVENT_HANDLER eventHandler,
const uintptr_t context
);
```

DRV_SST25VF016B_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockRead(const DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF016B_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockRead( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );
```

```
if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF016B memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF016B_BlockRead
(
const DRV_HANDLE handle,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF016B_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockWrite(DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

The flash address location which has to be written, must be erased before using the API DRV_SST25VF016B_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF016B_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockWrite( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:
```

```
            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF016B Flash where the write should begin. It can be any address of the flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

void DRV_SST25VF016B_BlockWrite
(
DRV_HANDLE handle,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

d) Media Interface Functions

DRV_SST25VF016B_GeometryGet Function

Returns the geometry of the device.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF016B_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF016B Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF016B_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;

// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total flash size should be 256k byte
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF016B_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF016B_MediaIsAttached Function

Returns the status of the media.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
bool DRV_SST25VF016B_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This API tells if the media is attached or not.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF016B_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF016B_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF016B_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf016b.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF016B_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF016B_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf016b.h

C

```
typedef enum {
  DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF016B_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF016B SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF016B_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF016B_CLIENT_STATUS Enumeration

Defines the client status.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef enum {
  DRV_SST25VF016B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF016B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF016B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF016B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF016B_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF016B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF016B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF016B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_SST25VF016B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF016B_EVENT_HANDLER Type

Pointer to a SST25VF016B SPI Flash Driver Event handler function.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef void (* DRV_SST25VF016B_EVENT_HANDLER)(DRV_SST25VF016B_BLOCK_EVENT event,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF016B SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF016B SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF016B_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF016B_BLOCK_EVENT event,
    DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF016B_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF016B_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used
PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position

uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF016B_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF016B_Initialize function.

DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF016B_INDEX_0 Macro

SPI Flash driver index definitions

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF016B SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF016B_Initialize and DRV_SST25VF016B_Open routines to identify the driver instance in use.

DRV_SST25VF016B_INDEX_1 Macro

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_INDEX_1 1
```

Description

This is macro DRV_SST25VF016B_INDEX_1.

SST25VF020B API

a) System Functions

	Name	Description
	DRV_SST25VF020B_Initialize	Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF020B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

b) Core Client Functions

	Name	Description
	DRV_SST25VF020B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_CommandStatus	Gets the current status of the command.
	DRV_SST25VF020B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic

c) Block Operation Functions

	Name	Description
	DRV_SST25VF020B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEraseWrite	Erase and Write blocks of data starting from a specified address in SST flash memory.

d) Media Interface Functions

	Name	Description
	DRV_SST25VF020B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF020B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF020B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF020B_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF020B_EVENT_HANDLER	Pointer to a SST25VF020B SPI Flash Driver Event handler function.
	DRV_SST25VF020B_INIT	Contains all the data necessary to initialize the SPI Flash device.
	DRV_SST25VF020B_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF020B_INDEX_0	SPI Flash driver index definitions.

	DRV_SST25VF020B_INDEX_1	This is macro DRV_SST25VF020B_INDEX_1.

Description

This section contains the SST25V020B Flash device API.

a) System Functions

DRV_SST25VF020B_Initialize Function

Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_MODULE_OBJ DRV_SST25VF020B_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF020B_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF020B SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF020B_INIT   SST25VF020BInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF020BInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF020BInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF020BInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF020BInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF020BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF020BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF020BInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF020BInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF020BInitData.queueSize = 5;

objectHandle = DRV_SST25VF020B_Initialize(DRV_SST25VF020B_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF020BInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SST25VF020B_Initialize
(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
);

DRV_SST25VF020B_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF020B_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF020B_Initialize
SYS_STATUS          status;


DRV_SST25VF020B_Deinitialize(object);

status = DRV_SST25VF020B_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF020B_Initialize

Function

void DRV_SST25VF020B_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF020B_Status Function

Gets the current status of the SPI Flash Driver module.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_STATUS DRV_SST25VF020B_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF020B_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF020B_Initialize
SYS_STATUS          SST25VF020BStatus;

SST25VF020BStatus = DRV_SST25VF020B_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF020BStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF020B_Initialize

Function

SYS_STATUS DRV_SST25VF020B_Status(SYS_MODULE_OBJ object)

DRV_SST25VF020B_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF020B_Initialize

while (true)
{
    DRV_SST25VF020B_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF020B_Initialize)

Function

void DRV_SST25VF020B_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF020B_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
DRV_SST25VF020B_CLIENT_STATUS DRV_SST25VF020B_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF020B_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF020B_Open
DRV_SST25VF020B_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF020B_ClientStatus(handle);
if(DRV_SST25VF020B_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_SST25VF020B_CLIENT_STATUS DRV_SST25VF020B_ClientStatus(DRV_HANDLE handle);

DRV_SST25VF020B_CommandStatus Function

Gets the current status of the command.

File

drv_sst25vf020b.h

C

```
DRV_SST25VF020B_COMMAND_STATUS DRV_SST25VF020B_CommandStatus(const DRV_HANDLE handle, const
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_SST25VF020B_COMMAND_STATUS value describing the current status of the buffer. Returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the buffer. The application must use this routine where the status of a scheduled buffer needs to polled on. The function may return DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in a case where the buffer handle has expired. A buffer handle expires when the internal buffer object is re-assigned to another erase, read or write request. It is recommended that this function be called regularly in order to track the buffer status correctly.

The application can alternatively register an event handler to receive write, read or erase operation completion events.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

Block command request must have been made using Erase, Read or Write APIs to get a valid command handle.

Example

```
DRV_HANDLE      sstOpenHandle;   // Returned from DRV_SST25VF020B_Open
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE  commandHandle;
DRV_SST25VF020B_BlockErase
                        (
                            sstOpenHandle,
                            &commandHandle,
                            0,
                            1
                        );

if(DRV_SST25VF020B_CommandStatus(sstOpenHandle, commandHandle) == DRV_SST25VF020B_COMMAND_COMPLETED );
{
    // do something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
commandHandle	A valid command handle, returned from Read/Write/Erase APIs.

Function

DRV_SST25VF020B_COMMAND_STATUS DRV_SST25VF020B_CommandStatus

(

const DRV_HANDLE handle,
const DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle
);

DRV_SST25VF020B_Close Function

Closes an opened-instance of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF020B_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF020B_Open

DRV_SST25VF020B_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF020B_Close(DRV_Handle handle);

DRV_SST25VF020B_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
DRV_HANDLE DRV_SST25VF020B_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF020B_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

The driver status becomes ready inside "DRV_SST25VF020B_Tasks" function. To make the SST Driver status ready and hence successfully "Open" the driver, "Task" routine need to be called periodically.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF020B_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF020B_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST25VF020B_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

);

c) Block Operation Functions

DRV_SST25VF020B_BlockErase Function

Erase the specified number of blocks in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockErase(const DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF020B_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF020B_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockErase( mySST25VF020BHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_ERASE_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF020B memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_SST25VF020B_BlockErase
(
const DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF020B_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF020B_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF020B_BlockEventHandlerSet( mySST25VF020BHandle,
```

```
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF020B_BlockRead( mySST25VF020BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF020B_BlockEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_SST25VF020B_EVENT_HANDLER eventHandler,

const uintptr_t context

);

DRV_SST25VF020B_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockRead(const DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF020B_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockRead( mySST25VF020BHandle, commandHandle,
                                      &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;
```

```
            case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

                // Error handling here.

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF020B memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF020B_BlockRead
(
const DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF020B_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockWrite(DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

The flash address location which has to be written, must be erased before using the API DRV_SST25VF020B_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF020B_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockWrite( mySST25VF020BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle

sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF020B Flash where the write should begin. It can be any address of the flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

void DRV_SST25VF020B_BlockWrite
(
DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF020B_BlockEraseWrite Function

Erase and Write blocks of data starting from a specified address in SST flash memory.

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockEraseWrite(DRV_HANDLE hClient, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Description

This function combines the step of erasing blocks of SST Flash and then writing the data. The application can use this function if it wants to avoid having to explicitly delete a block in order to update the bytes contained in the block.

This function schedules a non-blocking operation to erase and write blocks of data into SST flash. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF020B_EVENT_ERASE_ERROR event if the buffer was not processed successfully.

Remarks

Refer to drv_sst25vf020b.h for usage information.

Function

void DRV_SST25VF020B_BlockEraseWrite
(
const DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t writeBlockStart,
uint32_t nWriteBlock
)

d) Media Interface Functions

DRV_SST25VF020B_GeometryGet Function

Returns the geometry of the device.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF020B_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF020B Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF020B_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;

// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total flash size should be 256k byte
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF020B_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF020B_MediaIsAttached Function

Returns the status of the media.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
bool DRV_SST25VF020B_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This function determines whether or not the media is attached.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF020B_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF020B_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF020B_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf020b.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF020B_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF020B_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf020b.h

C

```
typedef enum {
```

```
  DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF020B_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF020B SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF020B_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF020B_CLIENT_STATUS Enumeration

Defines the client status.

File

drv_sst25vf020b.h

C

```
typedef enum {
  DRV_SST25VF020B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF020B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF020B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF020B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF020B_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF020B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF020B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF020B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_SST25VF020B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF020B_EVENT_HANDLER Type

Pointer to a SST25VF020B SPI Flash Driver Event handler function.

File

drv_sst25vf020b.h

C

```
typedef void (* DRV_SST25VF020B_EVENT_HANDLER)(DRV_SST25VF020B_BLOCK_EVENT event,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF020B SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF020B SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF020B_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF020B_BLOCK_EVENT event,
    DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF020B_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

File

drv_sst25vf020b.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
```

```
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF020B_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used
PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position
uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF020B_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF020B_Initialize function.

DRV_SST25VF020B_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_sst25vf020b.h

C

```
typedef enum {
  DRV_SST25VF020B_COMMAND_COMPLETED,
  DRV_SST25VF020B_COMMAND_QUEUED,
  DRV_SST25VF020B_COMMAND_IN_PROGRESS,
  DRV_SST25VF020B_COMMAND_ERROR_UNKNOWN
} DRV_SST25VF020B_COMMAND_STATUS;
```

Members

Members	Description
DRV_SST25VF020B_COMMAND_COMPLETED	Requested operation is completed
DRV_SST25VF020B_COMMAND_QUEUED	Scheduled but not started
DRV_SST25VF020B_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SST25VF020B_COMMAND_ERROR_UNKNOWN	Unknown Command

Description

SST Flash Driver Command Status

SST Flash Driver command Status

This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF020B_INDEX_0 Macro

SPI Flash driver index definitions.

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF020B SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF020B_Initialize and DRV_SST25VF020B_Open routines to identify the driver instance in use.

DRV_SST25VF020B_INDEX_1 Macro

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_INDEX_1 1
```

Description

This is macro DRV_SST25VF020B_INDEX_1.

SST25VF064C API

a) System Functions

	Name	Description
	DRV_SST25VF064C_Initialize	Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.
	DRV_SST25VF064C_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module.
	DRV_SST25VF064C_Status	Gets the current status of the SPI Flash Driver module.
	DRV_SST25VF064C_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR.

b) Core Client Functions

	Name	Description
	DRV_SST25VF064C_ClientStatus	Gets current client-specific status of the SPI Flash driver.
	DRV_SST25VF064C_Close	Closes an opened-instance of the SPI Flash driver.
	DRV_SST25VF064C_CommandStatus	Gets the current status of the command.

	DRV_SST25VF064C_Open	Opens the specified SPI Flash driver instance and returns a handle to it.

c) Block Operation Functions

	Name	Description
	DRV_SST25VF064C_BlockErase	Erase the specified number of blocks in Flash memory.
	DRV_SST25VF064C_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SST25VF064C_BlockRead	Reads blocks of data starting from the specified address in Flash memory.
	DRV_SST25VF064C_BlockWrite	Write blocks of data starting from a specified address in Flash memory.

d) Media Interface Functions

	Name	Description
	DRV_SST25VF064C_GeometryGet	Returns the geometry of the device.
	DRV_SST25VF064C_MediaIsAttached	Returns the status of the media.

e) Data Types and Constants

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF064C_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF064C_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF064C_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_SST25VF064C_EVENT_HANDLER	Pointer to a SST25VF064C SPI Flash Driver Event handler function.
	DRV_SST25VF064C_INIT	Contains all the data necessary to initialize the SPI Flash device.
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF064C_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF064C_INDEX_1	This is macro DRV_SST25VF064C_INDEX_1.

Description

a) System Functions

DRV_SST25VF064C_Initialize Function

Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.

File

drv_sst25vf064c.h

C

```
SYS_MODULE_OBJ DRV_SST25VF064C_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF064C_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF064C SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF064C_INIT   SST25VF064CInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF064CInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF064CInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF064CInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF064CInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF064CInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF064CInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF064CInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF064CInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF064CInitData.queueSize = 5;

objectHandle = DRV_SST25VF064C_Initialize(DRV_SST25VF064C_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF064CInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SST25VF064C_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_SST25VF064C_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF064C_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF064C_Initialize
SYS_STATUS          status;


DRV_SST25VF064C_Deinitialize(object);

status = DRV_SST25VF064C_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF064C_Initialize

Function

void DRV_SST25VF064C_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF064C_Status Function

Gets the current status of the SPI Flash Driver module.

File

drv_sst25vf064c.h

C

```
SYS_STATUS DRV_SST25VF064C_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF064C_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF064C_Initialize
SYS_STATUS          SST25VF064CStatus;

SST25VF064CStatus = DRV_SST25VF064C_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF064CStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF064C_Initialize

Function

SYS_STATUS DRV_SST25VF064C_Status(SYS_MODULE_OBJ object)

DRV_SST25VF064C_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF064C_Initialize

while (true)
{
    DRV_SST25VF064C_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF064C_Initialize)

Function

void DRV_SST25VF064C_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF064C_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

File

drv_sst25vf064c.h

C

```
DRV_SST25VF064C_CLIENT_STATUS DRV_SST25VF064C_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF064C_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF064C_Open
DRV_SST25VF064C_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF064C_ClientStatus(handle);
if(DRV_SST25VF064C_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_SST25VF064C_CLIENT_STATUS DRV_SST25VF064C_ClientStatus(DRV_HANDLE handle);

DRV_SST25VF064C_Close Function

Closes an opened-instance of the SPI Flash driver.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF064C_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF064C_Open

DRV_SST25VF064C_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF064C_Close(DRV_Handle handle);

DRV_SST25VF064C_CommandStatus Function

Gets the current status of the command.

File

drv_sst25vf064c.h

C

```
DRV_SST25VF064C_COMMAND_STATUS DRV_SST25VF064C_CommandStatus(const DRV_HANDLE handle, const
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_SST25VF064C_COMMAND_STATUS value describing the current status of the buffer. Returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the buffer. The application must use this routine where the status of a scheduled buffer needs to polled on. The function may return DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in a case where the buffer handle has expired. A buffer handle expires when the internal buffer object is re-assigned to another erase, read or write request. It is recommended that this function be called regularly in order to track the buffer status correctly.

The application can alternatively register an event handler to receive write, read or erase operation completion events.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

Block command request must have been made using Erase, Read or Write APIs to get a valid command handle.

Example

```
DRV_HANDLE      sstOpenHandle;   // Returned from DRV_SST25VF064C_Open
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE  commandHandle;
DRV_SST25VF064C_BlockErase
                        (
                            sstOpenHandle,
                            &commandHandle,
                            0,
                            1
                        );

if(DRV_SST25VF064C_CommandStatus(sstOpenHandle, commandHandle) == DRV_SST25VF064C_COMMAND_COMPLETED );
{
    // do something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
commandHandle	A valid command handle, returned from Read/Write/Erase APIs.

Function

DRV_SST25VF064C_COMMAND_STATUS DRV_SST25VF064C_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle

);

DRV_SST25VF064C_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

File

drv_sst25vf064c.h

C

```
DRV_HANDLE DRV_SST25VF064C_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF064C_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF064C_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF064C_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF064C_Open(DRV_SST25VF064C_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST25VF064C_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

);

c) Block Operation Functions

DRV_SST25VF064C_BlockErase Function

Erase the specified number of blocks in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockErase(const DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in Flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF064C_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF064C_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete Flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockErase( mySST25VF064CHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_ERASE_COMPLETE:
```

```
            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF064C memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_SST25VF064C_BlockErase
(
const DRV_HANDLE handle,
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF064C_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF064C_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;
```

```
uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF064C_BlockEventHandlerSet( mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF064C_BlockRead( mySST25VF064CHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF064C_BlockEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_SST25VF064C_EVENT_HANDLER eventHandler,

const uintptr_t context

);

DRV_SST25VF064C_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockRead(const DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from Flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF064C_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockRead( mySST25VF064CHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.
```

```
void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF064C memory from where the read should begin. It can be any address of the Flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF064C_BlockRead

(

const DRV_HANDLE handle,

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,

uint8_t *targetBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_SST25VF064C_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockWrite(DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into Flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL

- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete Flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

The Flash address location which has to be written, must be erased before using the API DRV_SST25VF064C_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF064C_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockWrite( mySST25VF064CHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
```

```
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF064C Flash where the write should begin. It can be any address of the Flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

```
void DRV_SST25VF064C_BlockWrite
(
    DRV_HANDLE handle,
    DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);
```

d) Media Interface Functions

DRV_SST25VF064C_GeometryGet Function

Returns the geometry of the device.

File

drv_sst25vf064c.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF064C_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF064C Flash:

- Media Property
- Number of Read/Write/Erase regions in the Flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF064C_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;
```

```
// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total Flash size should be 8 MB
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF064C_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF064C_MediaIsAttached Function

Returns the status of the media.

File

drv_sst25vf064c.h

C

```
bool DRV_SST25VF064C_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This function determines whether or not the media is attached.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF064C_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF064C_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf064c.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF064C_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF064C_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF064C_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF064C SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF064C_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF064C_CLIENT_STATUS Enumeration

Defines the client status.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF064C_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF064C_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF064C_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF064C_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF064C_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF064C_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF064C_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed

DRV_SST25VF064C_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF064C_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_COMMAND_COMPLETED,
  DRV_SST25VF064C_COMMAND_QUEUED,
  DRV_SST25VF064C_COMMAND_IN_PROGRESS,
  DRV_SST25VF064C_COMMAND_ERROR_UNKNOWN
} DRV_SST25VF064C_COMMAND_STATUS;
```

Members

Members	Description
DRV_SST25VF064C_COMMAND_COMPLETED	Requested operation is completed
DRV_SST25VF064C_COMMAND_QUEUED	Scheduled but not started
DRV_SST25VF064C_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SST25VF064C_COMMAND_ERROR_UNKNOWN	Unknown Command

Description

SST Flash Driver Command Status

SST Flash Driver command Status. This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_SST25VF064C_EVENT_HANDLER Type

Pointer to a SST25VF064C SPI Flash Driver Event handler function.

File

drv_sst25vf064c.h

C

```
typedef void (* DRV_SST25VF064C_EVENT_HANDLER)(DRV_SST25VF064C_BLOCK_EVENT event,
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF064C SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF064C SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF064C_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF064C_BLOCK_EVENT event,
    DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF064C_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

File

drv_sst25vf064c.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF064C_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used

PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position
uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF064C_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF064C_Initialize function.

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF064C_INDEX_0 Macro

SPI Flash driver index definitions.

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF064C SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF064C_Initialize and DRV_SST25VF064C_Open routines to identify the driver instance in use.

DRV_SST25VF064C_INDEX_1 Macro

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_INDEX_1 1
```

Description

This is macro DRV_SST25VF064C_INDEX_1.

Files

Files

Name	Description
drv_sst25vf016b.h	SPI Flash Driver Interface Definition
drv_sst25vf016b_config_template.h	SST25VF016B Driver Configuration Template.
drv_sst25vf020b.h	SPI Flash Driver Interface Definition
drv_sst25vf020b_config_template.h	SST25VF020B Driver Configuration Template.
drv_sst25vf064c.h	SPI Flash Driver Interface Definition
drv_sst25vf064c_config_template.h	SST25VF064C Driver Configuration Template.

Description

This section lists the source and header files used by the SPI Flash Driver Library.

drv_sst25vf016b.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF016B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF016B_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic

Functions

	Name	Description
	DRV_SST25VF016B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF016B_Initialize	Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF016B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_SST25VF016B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF016B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic

	DRV_SST25VF016B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF016B_INDEX_0	SPI Flash driver index definitions
	DRV_SST25VF016B_INDEX_1	This is macro DRV_SST25VF016B_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF016B_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic

Types

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF016B_EVENT_HANDLER	Pointer to a SST25VF016B SPI Flash Driver Event handler function. **Implementation:** Dynamic

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf016b.h

Company

Microchip Technology Inc.

drv_sst25vf016b_config_template.h

SST25VF016B Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF016B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF016B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF016B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_SST25VF016B_MODE	Determines whether the driver is implemented as static or dynamic
	DRV_SST25VF016B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF016B Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf016b_config_template.h

Company

Microchip Technology Inc.

drv_sst25vf020b.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF020B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF020B_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF020B_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.

Functions

	Name	Description
	DRV_SST25VF020B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEraseWrite	Erase and Write blocks of data starting from a specified address in SST flash memory.
	DRV_SST25VF020B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_CommandStatus	Gets the current status of the command.
	DRV_SST25VF020B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF020B_Initialize	Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF020B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_SST25VF020B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF020B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF020B_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF020B_INDEX_1	This is macro DRV_SST25VF020B_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF020B_INIT	Contains all the data necessary to initialize the SPI Flash device.

Types

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.

	DRV_SST25VF020B_EVENT_HANDLER	Pointer to a SST25VF020B SPI Flash Driver Event handler function.

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf020b.h

Company

Microchip Technology Inc.

drv_sst25vf020b_config_template.h

SST25VF020B Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF020B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF020B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF020B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF020B_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF020B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF020B Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf020b_config_template.h

Company

Microchip Technology Inc.

drv_sst25vf064c.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF064C_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF064C_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF064C_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.

Functions

	Name	Description
	DRV_SST25VF064C_BlockErase	Erase the specified number of blocks in Flash memory.
	DRV_SST25VF064C_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SST25VF064C_BlockRead	Reads blocks of data starting from the specified address in Flash memory.
	DRV_SST25VF064C_BlockWrite	Write blocks of data starting from a specified address in Flash memory.
	DRV_SST25VF064C_ClientStatus	Gets current client-specific status of the SPI Flash driver.
	DRV_SST25VF064C_Close	Closes an opened-instance of the SPI Flash driver.

	DRV_SST25VF064C_CommandStatus	Gets the current status of the command.
	DRV_SST25VF064C_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module.
	DRV_SST25VF064C_GeometryGet	Returns the geometry of the device.
	DRV_SST25VF064C_Initialize	Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.
	DRV_SST25VF064C_MediaIsAttached	Returns the status of the media.
	DRV_SST25VF064C_Open	Opens the specified SPI Flash driver instance and returns a handle to it.
	DRV_SST25VF064C_Status	Gets the current status of the SPI Flash Driver module.
	DRV_SST25VF064C_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR.

Macros

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF064C_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF064C_INDEX_1	This is macro DRV_SST25VF064C_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF064C_INIT	Contains all the data necessary to initialize the SPI Flash device.

Types

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF064C_EVENT_HANDLER	Pointer to a SST25VF064C SPI Flash Driver Event handler function.

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf064c.h

Company

Microchip Technology Inc.

drv_sst25vf064c_config_template.h

SST25VF064C Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF064C_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF064C_HARDWARE_HOLD_ENABLE	Specifies whether or not the hardware hold feature is enabled.
	DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies whether or not the hardware write protect feature is enabled.
	DRV_SST25VF064C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF064C_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF064C_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF064C Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf064c_config_template.h

Company

Microchip Technology Inc.

SPI PIC32WK IPF Flash Driver Library

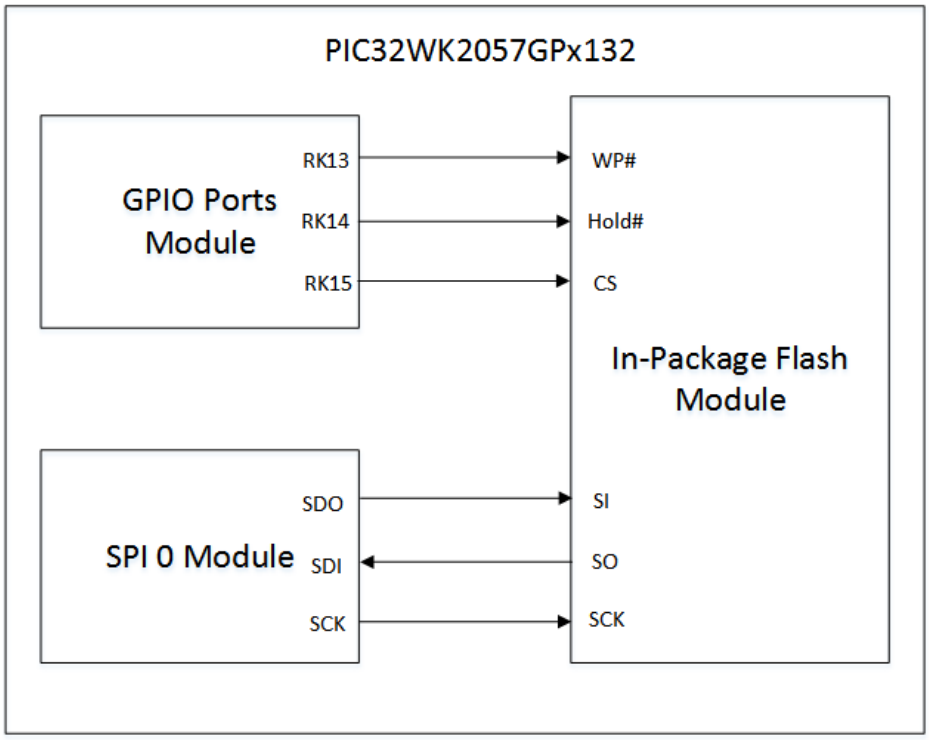
This section describes the Serial Peripheral Interface (SPI) Flash driver library for the PIC32WK IPF (in-package flash) module.

Introduction

This library provides an interface to manage the PIC32WK IPF module in different modes of operation.

Description

PIC32WK consists of an in-package flash (IPF) that is interfaced to the core using SPI, specifically the SPI0 instance. For more information, refer to the PIC32WK Silicon Data Sheet. The SPI module of the controller operates as a master device and the IPF module operates as a slave.



The PIC32WK IPF driver is dynamic in nature, therefore a single instance of it can support multiple clients that want to use the same flash. Multiple instances of the driver can be used when multiple flash devices are required to be part of the system. The SPI driver, which is used by the PIC32WK IPF driver, can be configured for use in either Polled or Interrupt mode.

Using the Library

This topic describes the basic architecture of the SPI PIC32WK IPF Flash Driver Library and provides information and examples about how to use it.

Description

Interface Header File: `drv_ipf.h`

The interface to the SPI PIC32WK IPF Flash Driver Library is defined in the `drv_ipf.h` header file. Any C language source (.c) file that uses the SPI PIC32WK IPF Flash Driver Library should include this header file.

Refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Source Files

The SPI PIC32WK IPF Flash Driver Library source files are provided in the `<install-dir>/framework/driver/spi_flash/pic32wk_ipf/src` folder. This folder may contain optional files and alternate

implementations. Refer to Configuring the Library for instructions about how to select optional features, and Building the Library for instructions about how to build the library.

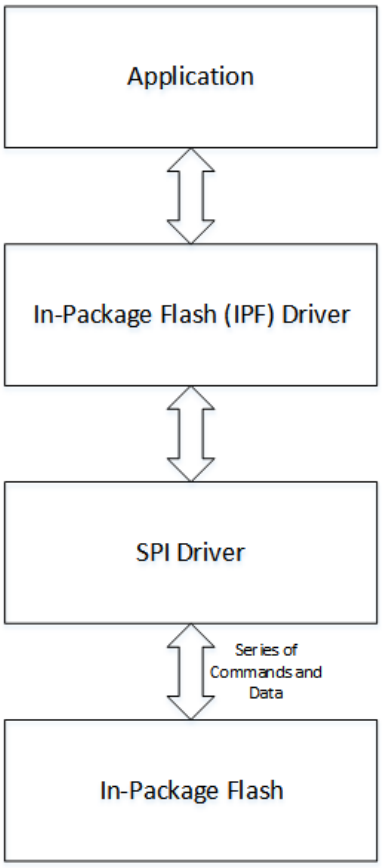
Abstraction Model

This section provides a low-level abstraction of the SPI PIC32WK IPF Flash Driver Library with a convenient C language interface. This topic describes how that abstraction is modeled in software.

Description

To perform a particular operation, the SPI PIC32WK IPF Flash Driver Library needs a specific set of commands to be given on its SPI interface along with the required address and data. The driver abstracts these requirements and provides simple APIs that can be used to perform Erase, Write, Read and memory protect operations. The SPI Driver is used for this purpose. The following layered diagram depicts the communication between different modules.

SPI PIC32WK IPF Flash Driver Library Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI PIC32WK IPF Flash Driver Library.

Library Interface Section	Description
System Functions	Accessed by the MPLAB Harmony system module and allow the driver to be initialized, de-initialized, and maintained.
Core Client Functions	Allow the application client to open and close the driver.
Block Operation Functions	Enable the Flash module to be erased, written, and read (to/from).
Media Interface Functions	Provide media status and the Flash geometry.
Memory Protection Functions	Functions protect or unprotect the required block of memory.

Pin Control Functions	Functions provide a means of controlling WP and Hold Pins.

How the Library Works

This topic describes the basic architecture of the SPI PIC32WK IPF Flash Driver Library and provides information and examples about its use.

Description

The library provides interfaces to support:

- System Initialization/Deinitialization
- Opening the Driver
- Block Operations

System Initialization/Deinitialization

This section provides information about initializing the system.

Description

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, the SPI PIC32WK IPF Flash Driver is initialized with the following configuration settings (either passed dynamically at runtime by using DRV_IPF_INIT or by using Initialization Overrides) that are supported or used by the IPF:

- Device-requested power state: one of the System Module Power States. For specific details please refer to "Data Types and Constants" in the Library Interface section.
- The SPI Driver Module Index, which is intended to be used to communicate with the IPF (for example, DRV_SPI_INDEX_0)
- Port Pins of the microcontroller to be used for Chip Select, Write Protection, and Hold operations on the IPF.
- Maximum Buffer Queue Size for that instance of the IPF.

The DRV_IPF_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface is used by the other system interfaces such as DRV_IPF_Deinitialize, DRV_ IPF _Status, and DRV_ IPF _Tasks.

Notes:

1. The system initialization and deinitialization settings affect only the instance of the peripheral that is being initialized or deinitialized.
2. As Hold, WP, and Chip select pins are internally routed, these are not configurable. Refer to the PIC32WK Silicon Data Sheet for more information.

Example:

```
// This code example shows the initialization of the In-Package Flash
// Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2,
// and 3 of PORTB are configured for the Hold pin, Write Protection pin, and
// the Chip Select pin, respectively. The maximum buffer queue size is set to 5.
DRV_IPF_INIT IPFInitData;
SYS_MODULE_OBJ objectHandle;
IPFInitData.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
IPFInitData.spiDriverModuleIndex = DRV_SPI_INDEX_0;
IPFInitData.holdPortChannel = PORT_CHANNEL_K;
IPFInitData.holdBitPosition = PORTS_BIT_POS_14;
IPFInitData.writeProtectPortChannel = PORT_CHANNEL_K;
IPFInitData.writeProtectBitPosition = PORTS_BIT_POS_13;
IPFInitData.chipSelectPortChannel = PORT_CHANNEL_K;
IPFInitData.chipSelectBitPosition = PORTS_BIT_POS_15;
IPFInitData.queueSize = 5;

objectHandle = DRV_IPF_Initialize(DRV_IPF_INDEX_0,(SYS_MODULE_INIT*)IPFInitData);

if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
// Handle error
}
```

Tasks Routine

The system will call DRV_IPF_Tasks, from SYS_Tasks. .

Opening the Driver

This section provides information about opening the driver.

Description

To use the SST Flash driver, the application must open the driver. Using the SST25VF020B as an example, this is done by calling the DRV_IPF_Open function. Calling this function with DRV_IO_INTENT_NONBLOCKING will cause the driver to be opened in non blocking mode. Then DRV_IPF_BlockErase, DRV_IPF_BlockWrite and DRV_IPF_BlockRead functions when called by this client will be non-blocking.

The client can also open the driver in Read-only mode (DRV_IO_INTENT_READ), Write-only mode (DRV_IO_INTENT_WRITE), and Exclusive mode (DRV_IO_INTENT_EXCLUSIVE). If the driver has been opened exclusively by a client, it cannot be opened again by another client. If successful, the DRV_IPF_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_IPF_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return and invalid handle in other (error) cases as well.

The following code shows an example of the driver being opened in different modes.

```
DRV_HANDLE ipfHandle1, ipfHandle2;
/* Client 1 opens the IPF driver in non blocking mode */
ipfHandle1 = DRV_IPF_Open(DRV_IPF_INDEX_0, DRV_IO_INTENT_NONBLOCKING);
/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == ipfHandle1)
{
/* The driver was not opened successfully. The client
* can try opening it again */
}
/* Client 2 opens the IPF driver in Exclusive Write only mode */
ipfHandle2 = DRV_IPF_Open(DRV_IPF_INDEX_0, DRV_IO_INTENT_WRITE | DRV_IO_INTENT_EXCLUSIVE);
/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == ipfHandle2)
{
/* The driver was not opened successfully. The client
* can try opening it again */
}
```

Block Operations

This section provides information about block operations.

Description

This driver provides simple client interfaces to Erase, Write, and Read the IPF in blocks. A block is the unit to represent minimum amount of data that can be erased, written, or read. Block size may differ for Erase, Write, and Read operations. The DRV_IPF_GeometryGet function can be used to determine the different block sizes for the driver.

The DRV_IPF_BlockErase, DRV_IPF_BlockWrite, and DRV_IPF_BlockRead functions are used to erase, write, and read the data to/from IPF. These functions are always non-blocking. All of these functions follow a standard queue model to read, write, and erase. When any of these functions are called (i.e., a block request is made), the request is queued. The size of the queue is determined by the queueSize member of the DRV_IPF_INIT data structure. All of the requests in the queue are executed by the DRV_IPF_Tasks function one-by-one.

When the driver adds a request to the queue, it returns a buffer handle. This handle allows the client to track the request as it progresses through the queue. The buffer handle expires when the event associated with the buffer completes. The driver provides driver events (DRV_IPF_BLOCK_EVENT) that indicate termination of the buffer requests.

For a simple Block Data Operation, perform the following steps :

1. The system should have completed necessary initialization of the SPI Driver and the IPF Driver, and the DRV_IPF_Tasks function should be running in a polled environment.
2. The DRV_SPI_Tasks function should be running in either a polled environment or an interrupt environment.
3. Open the driver using DRV_IPF_Open with the necessary intent.
4. Set an event handler callback using the function DRV_IPF_BlockEventHandlerSet.
5. Request for block operations using the functions, DRV_IPF_BlockErase, DRV_IPF_BlockWrite, and DRV_IPF_BlockRead, with the appropriate parameters.
6. Wait for event handler callback to occur and check the status of the block operation using the callback function parameter of type DRV_IPF_BLOCK_EVENT.
7. The client will be able to close the driver using the function, DRV_IPF_Close, when required.

Example:

```
/* This code example shows usage of the block operations
* on the PIC32WK IPF */
DRV_HANDLE ipfHandle1;
uint8_t myData1[10], myData2[10];
DRV_IPF_BLOCK_COMMAND_HANDLE blockHandle1, blockHandle2, blockHandle3;

/* The driver is opened for read-write in Exclusive mode */
```

```
ipfHandle1 = DRV_IPF_Open(DRV_IPF_INDEX_0,
DRV_IO_INTENT_READWRITE | DRV_IO_INTENT_EXCLUSIVE);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == ipfHandle1)
{
 /* The driver could not be opened successfully */
}

/* Register a Buffer Event Handler with IPF driver.
* This event handler function will be called whenever
* there is a buffer event. An application defined
* context can also be specified. This is returned when
* the event handler is called.
* */
DRV_IPF_BlockEventHandlerSet(sstHandle1,APP_IPFBufferEventHandler, NULL);

/* Request for all the three block operations one by one */
/* first block API to erase 1 block of the flash starting from address 0x0, each block is of 4kbyte */
DRV_IPF_BlockErase(ipfHandle1, &blockHandle1, 0x0, 1);

/* 2nd block API to write myData1 in the first 10 locations of the flash */
DRV_SST25VF020B_BlockWrite(ipfHandle1, &blockHandle2, &myData1[0], 0x0, 10);

/* 3rd block API to read the first 10 locations of the flash into myData2 */
DRV_SST25VF020B_BlockRead(ipfHandle1, &blockHandle3, &myData2[0], 0x0, 10);

/* This is the Driver Event Handler */
void APP_IPFBufferEventHandler(DRV_SST25VF020B_BLOCK_EVENT event, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE
blockHandle, uintptr_t contextHandle)
{
 switch(event)
 {
 case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:
 if ( blockHandle == blockHandle3)
 {
 /* This means the data was read */
 /* Do data verification/processing */
 }
 break;
 case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:
 /* Error handling here. */
 break;
 default:
 break;
 }
```

Configuring the Library

Use this section for the drivers and system services and middleware. This section will contain any related configuration macros imported into the project from the companion `<library>_config_template.h` file into this topic, if applicable.

Description

The SPI PIC32WK IPF Flash Driver Library requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. For more details, refer to Applications Help.

Building the Library

This section lists the files that are available in the SPI PIC32WK IPF Flash Driver Library.

Description

This section list the files that are available in the `\src` folder of the SPI PIC32WK IPF Flash Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is

`<install-dir>/framework/driver/spi_flash/pic32wk_ipf`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ipf.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ipf.c`	Basic SPI PIC32WK IPF Flash Driver implementation file.
`/src/dynamic/drv_ipf_fs.c`	File system functions used by the driver API.
`/src/drv_ipf_prot.c`	Protocol implementation used by the driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SPI PIC32WK IPF Flash Driver Library depends on the following modules:

- Clock System Service Library

Optional Dependencies

- DMA System Service Library (used when operating in DMA mode)
- Interrupt System Service Library (used when task is running in Interrupt mode)

Library Interface

a) System Initialization Functions

	Name	Description
	DRV_IPF_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_IPF_Initialize	Initializes the IPF SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_IPF_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_IPF_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_IPF_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_IPF_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_IPF_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic

c) Other Functions

	Name	Description
	DRV_IPF_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic

	DRV_IPF_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_IPF_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_IPF_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_IPF_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_IPF_HoldAssert	Asserts the Hold pin for flash. **Implementation:** Dynamic
	DRV_IPF_HoldDeAssert	Deasserts the Hold pin for flash. **Implementation:** Dynamic
	DRV_IPF_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_IPF_ProtectMemoryVolatile	Protects the memory block to which the given memory address belongs **Implementation:** Dynamic
	DRV_IPF_ReadBlockProtectionStatus	Reads the content of Block Protection Register which belongs to In-Package flash. **Implementation:** Dynamic
	DRV_IPF_UnProtectMemoryVolatile	Un-protects the memory block to which the given memory address belongs **Implementation:** Dynamic
	DRV_IPF_WPAssert	Asserts the WP pin for flash. **Implementation:** Dynamic
	DRV_IPF_WPDeAssert	Deasserts the WP pin for flash. **Implementation:** Dynamic

d) Data Types and Constants

	Name	Description
	DRV_IPF_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_IPF_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_IPF_BLOCK_OPERATION	Lists the different operations that IPF driver can do.
	DRV_IPF_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_IPF_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_IPF_EVENT_HANDLER	Pointer to a IPF SPI Flash Driver Event handler function. **Implementation:** Dynamic
	DRV_IPF_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic
	DRV_IPF_PROT_MODE	Lists the different memory protection modes.
	_DRV_IPF_H	This is macro _DRV_IPF_H.
	DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_IPF_INDEX_0	SPI Flash driver index definitions
	_DRV_IPF_CONFIG_TEMPLATE_H	This is macro _DRV_IPF_CONFIG_TEMPLATE_H.
	DRV_IPF_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to the hardware instance.
	DRV_IPF_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_IPF_MODE	Determines whether the driver is implemented as static or dynamic

Description

This section describes the API functions of the SPI PIC32WK IPF Flash Driver library.

Refer to each section for a detailed description.

a) System Initialization Functions

DRV_IPF_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_IPF_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_IPF_Initialize
SYS_STATUS          status;


DRV_IPF_Deinitialize(object);

status = DRV_IPF_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_IPF_Initialize

Function

void DRV_IPF_Deinitialize(SYS_MODULE_OBJ object)

DRV_IPF_Initialize Function

Initializes the IPF SPI Flash Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_ipf.h

C

```
SYS_MODULE_OBJ DRV_IPF_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_IPF_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the IPF SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_IPF_INIT   IPFInitData;
SYS_MODULE_OBJ      objectHandle;

IPFInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
IPFInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
IPFInitData.holdPortChannel       = PORT_CHANNEL_B;
IPFInitData.holdBitPosition       = PORTS_BIT_POS_1;
IPFInitData.writeProtectPortChannel = PORT_CHANNEL_B;
IPFInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
IPFInitData.chipSelectPortChannel = PORT_CHANNEL_F;
IPFInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
IPFInitData.queueSize = 5;

objectHandle = DRV_IPF_Initialize(DRV_IPF_INDEX_0,
                                  (SYS_MODULE_INIT*)IPFInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_IPF_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_IPF_Status Function

Gets the current status of the SPI Flash Driver module.

Implementation: Dynamic

File

drv_ipf.h

C

```
SYS_STATUS DRV_IPF_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_IPF_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_IPF_Initialize
SYS_STATUS          IPFStatus;

IPFStatus = DRV_IPF_Status(object);
else if (SYS_STATUS_ERROR >= IPFStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_IPF_Initialize

Function

SYS_STATUS DRV_IPF_Status(SYS_MODULE_OBJ object)

DRV_IPF_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_IPF_Initialize

while (true)
{
    DRV_IPF_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_IPF_Initialize)

Function

void DRV_IPF_Tasks (SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_IPF_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

Implementation: Dynamic

File

drv_ipf.h

C

```
DRV_IPF_CLIENT_STATUS DRV_IPF_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_IPF_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_IPF_Initialize function must have been called.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_IPF_Open
DRV_IPF_CLIENT_STATUS     clientStatus;

clientStatus = DRV_IPF_ClientStatus(handle);
if(DRV_IPF_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_IPF_CLIENT_STATUS DRV_IPF_ClientStatus(DRV_HANDLE handle);

DRV_IPF_Close Function

Closes an opened-instance of the SPI Flash driver.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_IPF_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_IPF_Open

DRV_IPF_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_IPF_Close(DRV_Handle handle);

DRV_IPF_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ipf.h

C

```
DRV_HANDLE DRV_IPF_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_IPF_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

The driver status becomes ready inside "DRV_IPF_Tasks" function. To make the SST Driver status ready and hence successfully "Open" the driver, "Task" routine need to be called periodically.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_IPF_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_IPF_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_IPF_Open(DRV_IPF_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_IPF_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
);

c) Other Functions

DRV_IPF_BlockErase Function

Erase the specified number of blocks in Flash memory.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_BlockErase(const DRV_HANDLE handle, DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle, uint32_t
blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_IPF_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_IPF_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_IPF_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver

DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_BlockErase( myIPFHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_IPF_EVENT_ERASE_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_IPF_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in IPF memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_IPF_BlockErase

(

const DRV_HANDLE handle,

DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,
uint32_t nBlock
);

DRV_IPF_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_IPF_EVENT_HANDLER eventHandler, const
uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver. This is done once.

DRV_IPF_BlockEventHandlerSet( myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj );

DRV_IPF_BlockRead( myIPFHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;
```

```
    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_IPF_BlockEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_IPF_EVENT_HANDLER eventHandler,

const uintptr_t context

);

DRV_IPF_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_BlockRead(const DRV_HANDLE handle, DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle, uint8_t *
targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE

event if the buffer was processed successfully of DRV_IPF_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_IPF_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = IPF_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver

DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_BlockRead( myIPFHandle, commandHandle,
                                      &myBuffer, blockStart, nBlock );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed

blockStart	Start block address in IPF memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_IPF_BlockRead
(
const DRV_HANDLE handle,
DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_IPF_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_BlockWrite(DRV_HANDLE handle, DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle, uint8_t *
sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_IPF_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_IPF_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_IPF_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_IPF_Open call.

The flash address location which has to be written, must be erased before using the API DRV_IPF_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = IPF_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
```

```
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver

DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_BlockWrite( myIPFHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of IPF Flash where the write should begin. It can be any address of the flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

void DRV_IPF_BlockWrite

(

DRV_HANDLE handle,

DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,

uint8_t *sourceBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_IPF_GeometryGet Function

Returns the geometry of the device.

Implementation: Dynamic

File

drv_ipf.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_IPF_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the IPF Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_IPF_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;

// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total flash size should be 256k byte
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_IPF_GeometryGet(DRV_HANDLE handle);

DRV_IPF_HoldAssert Function

Asserts the Hold pin for flash.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_HoldAssert();
```

Returns

None.

Description

This API is used to assert the Hold pin of the in-package flash.

Remarks

The Hold GPIO is fixed in case of PIC32WK devices.

Preconditions

None.

Example

```
DRV_IPF_HoldAssert();
```

Function

void DRV_IPF_HoldAssert();

DRV_IPF_HoldDeAssert Function

Deasserts the Hold pin for flash.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_HoldDeAssert();
```

Returns

None.

Description

This API is used to deassert the Hold pin of the in-package flash.

Remarks

The Hold GPIO is fixed in case of PIC32WK devices.

Preconditions

None.

Example

```
DRV_IPF_HoldDeAssert();
```

Function

void DRV_IPF_HoldDeAssert();

DRV_IPF_MediaIsAttached Function

Returns the status of the media.

Implementation: Dynamic

File

drv_ipf.h

C

```
bool DRV_IPF_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This API tells if the media is attached or not.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_IPF_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_IPF_MediaIsAttached(DRV_HANDLE handle);

DRV_IPF_ProtectMemoryVolatile Function

Protects the memory block to which the given memory address belongs

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_ProtectMemoryVolatile(DRV_HANDLE clientHandle, DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uintptr_t memAddress, DRV_IPF_PROT_MODE protMode);
```

Returns

None.

Description

This API is used to protect the memory block to which a given memory address belongs. Both read and write protection mode is supported. The memory will be protected until the next power cycle.

Remarks

Only the selected blocks can be read protected, which is as per the in-package flash specification.

Preconditions

In-package flash driver open function must be called and a valid client handle must be available.

Example

```
uintptr_t memAddr = IPF_ADDRESS_PROTECT;
DRV_IPF_PROT_MODE    protMode = DRV_IPF_WRITE_PROTECT;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver
```

```
DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_ProtectMemoryVolatile( myIPFHandle, commandHandle,
                                        memAddr, protMode );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the memory protection is complete.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
clientHandle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
memAddress	Memory address which belongs to the memory block which needs to be protected
protMode	Read or write protect mode. If a block needs to be protected for both read and write, then both enum values can be ORed and passed to the function.

Function

void DRV_IPF_ProtectMemoryVolatile
(
    DRV_HANDLE clientHandle,
    DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uintptr_t memAddress,
    DRV_IPF_PROT_MODE protMode
);

DRV_IPF_ReadBlockProtectionStatus Function

Reads the content of Block Protection Register which belongs to In-Package flash.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_ReadBlockProtectionStatus(DRV_HANDLE clientHandle, DRV_IPF_BLOCK_COMMAND_HANDLE *
```

```
commandHandle, uint8_t * buffer);
```

Returns

None.

Description

This API is read the current contents of the block protection register in in-package flash and fills the buffer passed by the client.

Remarks

The block protection word is 6-bytes wide.

Preconditions

In-package flash driver open function must be called and a valid client handle must be available.

Example

```
uint8_t buf[6] = {0,};
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver

DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_ReadBlockProtectionStatus( myIPFHandle, commandHandle,
                                        buf );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the BPR read is complete.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
clientHandle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
buffer	pointer to a buffer to which the block protection status has to be updated

Function

void DRV_IPF_ReadBlockProtectionStatus
(
DRV_HANDLE clientHandle,
DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * buffer
);

DRV_IPF_UnProtectMemoryVolatile Function

Un-protects the memory block to which the given memory address belongs

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_UnProtectMemoryVolatile(DRV_HANDLE clientHandle, DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uintptr_t memAddress, DRV_IPF_PROT_MODE protMode);
```

Returns

None.

Description

This API is used to un-protect the memory block to which a given memory address belongs. Both read and write protection mode is supported. The memory will be protected until the next power cycle.

Remarks

If the memory block a client is trying to unprotect, is protected by some other client, then memory unprotection will not executed. The function will return without unprotecting.

Preconditions

In-package flash driver open function must be called and a valid client handle must be available.

Example

```
uintptr_t memAddr = IPF_ADDRESS_UNPROTECT;
DRV_IPF_PROT_MODE    protMode = DRV_IPF_WRITE_PROTECT;
DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myIPFHandle is the handle returned
// by the DRV_IPF_Open function.

// Client registers an event handler with driver

DRV_IPF_BlockEventHandlerSet(myIPFHandle,
                APP_IPFEventHandler, (uintptr_t)&myAppObj);

DRV_IPF_UnProtectMemoryVolatile( myIPFHandle, commandHandle,
                                        memAddr, protMode );

if(DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_IPFEventHandler(DRV_IPF_BLOCK_EVENT event,
        DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
        switch(event)
        {
            case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

                // This means the memory unprotection is complete.
                break;

            case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:

                // Error handling here.

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
clientHandle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
memAddress	Memory address which belongs to the memory block which needs to be un-protected
protMode	Read or write protect mode. If a block needs to be un-protected for both read and write, then both enum values can be ORed and passed to the function.

Function

```
void DRV_IPF_UnProtectMemoryVolatile
(
    DRV_HANDLE clientHandle,
    DRV_IPF_BLOCK_COMMAND_HANDLE * commandHandle,
uintptr_t memAddress,
    DRV_IPF_PROT_MODE protMode
);
```

DRV_IPF_WPAssert Function

Asserts the WP pin for flash.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_WPAssert();
```

Returns

None.

Description

This API is used to assert the Write Protect (WP) pin of the in-package flash.

Remarks

The Write Protection GPIO is fixed in case of PIC32WK devices.

Preconditions

None.

Example

```
DRV_IPF_WPAssert();
```

Function

void DRV_IPF_WPAssert();

DRV_IPF_WPDeAssert Function

Deasserts the WP pin for flash.

Implementation: Dynamic

File

drv_ipf.h

C

```
void DRV_IPF_WPDeAssert();
```

Returns

None.

Description

This API is used to deassert the Write Protect (WP) pin of the in-package flash.

Remarks

The Write Protection GPIO is fixed in case of PIC32WK devices.

Preconditions

None.

Example

```
DRV_IPF_WPDeAssert();
```

Function

void DRV_IPF_WPAssert();

d) Data Types and Constants

DRV_IPF_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_ipf.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_IPF_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_IPF_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_ipf.h

C

```
typedef enum {
  DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_IPF_EVENT_BLOCK_COMMAND_ERROR
} DRV_IPF_BLOCK_EVENT;
```

Members

Members	Description
DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_IPF_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

IPF SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_IPF_BlockEventHandlerSet function when a block request is completed.

DRV_IPF_BLOCK_OPERATION Enumeration

Lists the different operations that IPF driver can do.

File

drv_ipf.h

C

```
typedef enum {
  DRV_IPF_BLOCK_READ,
  DRV_IPF_BLOCK_WRITE,
  DRV_IPF_BLOCK_ERASE,
  DRV_IPF_HW_BLOCK_PROT,
  DRV_IPF_HW_BLOCK_UNPROT,
  DRV_IPF_READ_HW_BLOCK_PROT
} DRV_IPF_BLOCK_OPERATION;
```

Members

Members	Description
DRV_IPF_BLOCK_READ	Block Read
DRV_IPF_BLOCK_WRITE	Block Write
DRV_IPF_BLOCK_ERASE	Block Erase
DRV_IPF_HW_BLOCK_PROT	Hardware Block Protection
DRV_IPF_HW_BLOCK_UNPROT	Hardware Block Un-Protection
DRV_IPF_READ_HW_BLOCK_PROT	Read HW Block Protection Status

Description

IPF Driver Operations

This enumeration lists the different operations that IPF driver can do.

Remarks

None.

DRV_IPF_CLIENT_STATUS Enumeration

Defines the client status.

Implementation: Dynamic

File

drv_ipf.h

C

```
typedef enum {
  DRV_IPF_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_IPF_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_IPF_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_IPF_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_IPF_CLIENT_STATUS;
```

Members

Members	Description
DRV_IPF_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_IPF_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_IPF_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_IPF_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_IPF_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_ipf.h

C

```
typedef enum {
  DRV_IPF_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_IPF_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_IPF_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_IPF_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_IPF_COMMAND_STATUS;
```

Members

Members	Description
DRV_IPF_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_IPF_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_IPF_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_IPF_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

IPF Driver Command Status

IPF Driver command Status

This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_IPF_EVENT_HANDLER Type

Pointer to a IPF SPI Flash Driver Event handler function.

Implementation: Dynamic

File

drv_ipf.h

C

```
typedef void (* DRV_IPF_EVENT_HANDLER)(DRV_IPF_BLOCK_EVENT event, DRV_IPF_BLOCK_COMMAND_HANDLE
commandHandle, uintptr_t context);
```

Returns

None.

Description

IPF SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the IPF SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_IPF_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_IPF_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_IPF_BLOCK_EVENT event,
    DRV_IPF_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_IPF_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_IPF_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event

commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_IPF_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

Implementation: Dynamic

File

drv_ipf.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_IPF_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used
PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position
uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_IPF_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_IPF_Initialize function.

DRV_IPF_PROT_MODE Enumeration

Lists the different memory protection modes.

File

drv_ipf.h

C

```
typedef enum {
  DRV_IPF_WRITE_PROTECT = 1,
  DRV_IPF_READ_PROTECT
} DRV_IPF_PROT_MODE;
```

Members

Members	Description
DRV_IPF_WRITE_PROTECT = 1	Write Protect
DRV_IPF_READ_PROTECT	Read Protect

Description

IPF Driver memory protection modes

This enumeration lists the different memory protection modes.

Remarks

None.

_DRV_IPF_H Macro

File

drv_ipf.h

C

```
#define _DRV_IPF_H
```

Description

This is macro _DRV_IPF_H.

DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_ipf.h

C

```
#define DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_IPF_INDEX_0 Macro

SPI Flash driver index definitions

File

drv_ipf.h

C

```
#define DRV_IPF_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide IPF SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_IPF_Initialize and DRV_IPF_Open routines to identify the driver instance in use.

_DRV_IPF_CONFIG_TEMPLATE_H Macro

File

drv_ipf_config_template.h

C

```
#define _DRV_IPF_CONFIG_TEMPLATE_H
```

Description

This is macro _DRV_IPF_CONFIG_TEMPLATE_H.

DRV_IPF_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to the hardware instance.

File

drv_ipf_config_template.h

C

```
#define DRV_IPF_CLIENTS_NUMBER 4
```

Description

IPF Client Count Configuration

Sets up the maximum number of clients that can be connected to the hardware instance. So if IPF will be accessed by 2 clients then this number should be 2. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space.

Remarks

None.

DRV_IPF_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ipf_config_template.h

C

```
#define DRV_IPF_INSTANCES_NUMBER 1
```

Description

IPF driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of IPF modules that are needed by the application. Hardware Instance support consumes RAM memory space.

Remarks

As PIC32WK has only 1 instance of IPF, this macro is always set to 1.

DRV_IPF_MODE Macro

Determines whether the driver is implemented as static or dynamic

File

drv_ipf_config_template.h

C

```
#define DRV_IPF_MODE DYNAMIC
```

Description

IPF mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

Files

Files

Name	Description
drv_ipf.h	SPI Flash Driver Interface Definition
drv_ipf_config_template.h	IPF Driver Configuration Template.

Description

This section lists the source and header files used by the SPI PIC32WK IPF Flash Driver Library.

drv_ipf.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_IPF_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_IPF_BLOCK_OPERATION	Lists the different operations that IPF driver can do.
	DRV_IPF_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_IPF_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_IPF_PROT_MODE	Lists the different memory protection modes.

Functions

	Name	Description
	DRV_IPF_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_IPF_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_IPF_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_IPF_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_IPF_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_IPF_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_IPF_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_IPF_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_IPF_HoldAssert	Asserts the Hold pin for flash. **Implementation:** Dynamic
	DRV_IPF_HoldDeAssert	Deasserts the Hold pin for flash. **Implementation:** Dynamic
	DRV_IPF_Initialize	Initializes the IPF SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_IPF_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_IPF_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_IPF_ProtectMemoryVolatile	Protects the memory block to which the given memory address belongs **Implementation:** Dynamic
	DRV_IPF_ReadBlockProtectionStatus	Reads the content of Block Protection Register which belongs to In-Package flash. **Implementation:** Dynamic

	DRV_IPF_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_IPF_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic
	DRV_IPF_UnProtectMemoryVolatile	Un-protects the memory block to which the given memory address belongs **Implementation:** Dynamic
	DRV_IPF_WPAssert	Asserts the WP pin for flash. **Implementation:** Dynamic
	DRV_IPF_WPDeAssert	Deasserts the WP pin for flash. **Implementation:** Dynamic

Macros

	Name	Description
	_DRV_IPF_H	This is macro _DRV_IPF_H.
	DRV_IPF_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_IPF_INDEX_0	SPI Flash driver index definitions

Structures

	Name	Description
	DRV_IPF_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic

Types

	Name	Description
	DRV_IPF_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_IPF_EVENT_HANDLER	Pointer to a IPF SPI Flash Driver Event handler function. **Implementation:** Dynamic

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_IPF.h

Company

Microchip Technology Inc.

drv_ipf_config_template.h

IPF Driver Configuration Template.

Macros

	Name	Description
	_DRV_IPF_CONFIG_TEMPLATE_H	This is macro _DRV_IPF_CONFIG_TEMPLATE_H.
	DRV_IPF_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to the hardware instance.
	DRV_IPF_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_IPF_MODE	Determines whether the driver is implemented as static or dynamic

Description

IPF Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ipf_config_template.h

Company

Microchip Technology Inc.

SQI Driver Library

Introduction

This library provides an interface to manage the Serial Quad Interface (SQI) module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony Serial Quad Interface (SQI) Driver provides a high-level interface to the SQI peripherals on Microchip's PIC32 microcontrollers. The SQI Driver includes the following features:

- Provides application ready routines to read and write data to an SQI peripheral
- Supports Single, Dual, and Quad Lane modes
- Supports Single Data Rate (SDR)
- Supports Interrupt mode operation only
- Supports multi-client operation
- Provides data transfer events
- Supports non-blocking mode operation only
- Features thread-safe functions for use in RTOS applications
- Uses the SQI module's internal DMA Controller for transfers

Using the Library

This topic describes the basic architecture of the SQI Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_sqi.h`

The interface to the SQI Driver Library is defined in the `drv_sqi.h` header file. Any C language source (`.c`) file that uses the SQI Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

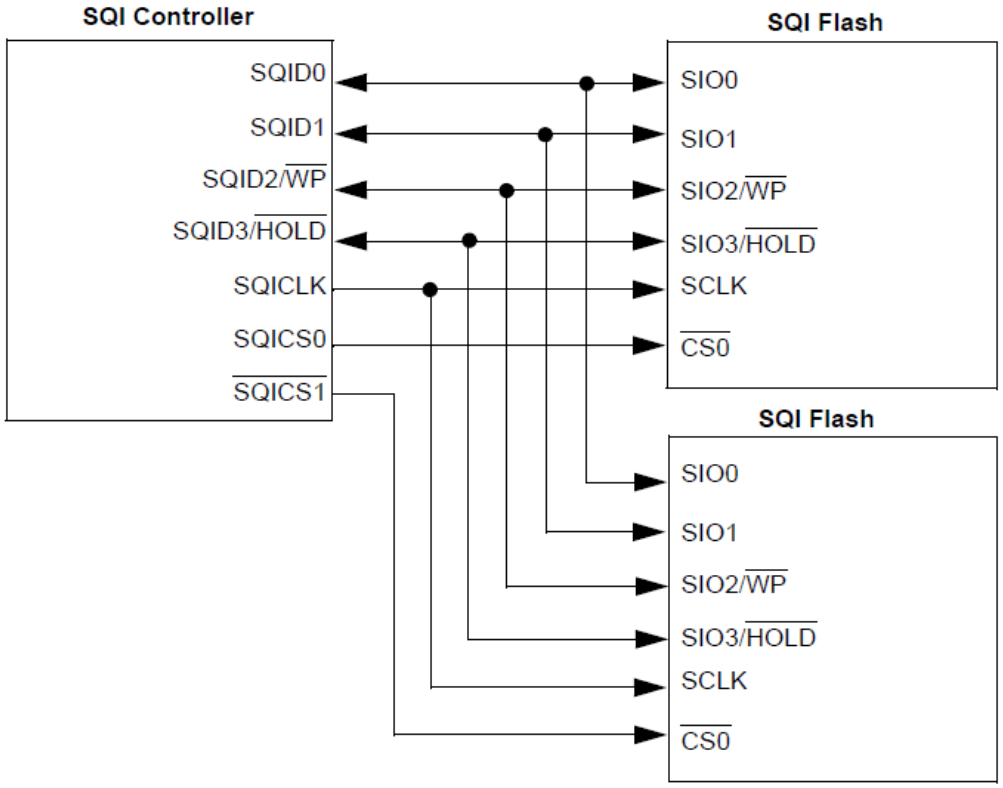
Abstraction Model

This library provides a low-level abstraction of the SQI Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SQI Driver Library supports up to two Chip Select lines. The following diagram shows the high SQI Driver and the SQI Flash sub-system

SQI Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SQI module.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the modes that are supported for your device.

System Functions

Provides information on the system functions provided in the SQI Driver Library.

Description

SQI Driver Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the SQI would be initialized with the following configuration settings (either passed dynamically at run time using DRV_SQI_INIT or by using initialization overrides) that are supported by the specific SQI Controller hardware:

- SQI Peripheral ID - Identifies the SQI Peripheral ID to be used
- Interrupt Source - The interrupt source associated with the SQI Controller
- Enabled Devices - Number of devices to enable
- Device Configuration - This configuration is per enabled device. A maximum of two devices are supported. The following configurations are allowed per device:
- Clock Divider Value - Clock divider value to be used
- SPI Mode of Operation - SPI mode of operation to be used for this device
- LSB First - Send or receive least significant bit of a byte first

The DRV_SQI_Initialize function configures and initializes the SQI controller using the configuration information provided. It returns an object handle of the type SYS_MODULE_OBJ. This object handle would be used by other system interfaces such as DRV_SQI_Status, DRV_SQI_Tasks and DRV_SQI_Deinitialize.

Example:

```
/* SQI Driver Initialization Data */
const DRV_SQI_INIT drvSqiInit =
{
    .sqiId = SQI_ID_0,
    .interruptSource = INT_SOURCE_SQI1,
    .enabledDevices = DRV_SQI_ENABLE_DEVICE_1,
    .clockDivider = DRV_SQI_CLK_DIV_1,
    .devCfg[0].spiMode = DRV_SQI_SPI_MODE_0,
    .devCfg[0].lsbFirst = false,
};

/* Initialize the SQI Driver */
sysObj.drvSqi = DRV_SQI_Initialize(DRV_SQI_INDEX_0, (SYS_MODULE_INIT *)&drvSqiInit);
```

SQI Driver Task Routine

The SQI driver data transfers are interrupt driven. The data transfer request from the client results in the SQI driver kick starting the transfer if there is no transfer in progress otherwise the request is added to the driver queue. The SQI interrupt handler is responsible for invoking the DRV_SQI_Tasks, which maintains the driver state machine. The task routine checks if the current request is complete and if there is another data transfer request queued, then it kick starts the processing of the request.

SQI Driver Status

DRV_SQI_Status returns the current status of the SQI driver module and is called by the Harmony System. The application may not find the need to call this function directly.

Example:

```
SYS_MODULE_OBJ object;
// Returned from DRV_SQI_Initialize
SYS_STATUS sqiStatus;

sqiStatus = DRV_SQI_Status(object);
if (SYS_STATUS_ERROR >= sqiStatus)
{
    // Handle error
}
```

Client Core Functions

Provides information on the client core functions provided in the SQI Driver Library

Description

Opening the Driver

For the application to start using an instance of the module, it must call the DRV_SQI_Open function repeatedly until a valid handle is returned by the driver. This provides the configuration required to open the SQI instance for operation.

For the various options available for I/O INTENT please refer to "Data Types and Constants" in the Library Interface section.

Example:

```
sqiHandle = DRV_SQI_Open (0, DRV_IO_INTENT_READWRITE);
if (sqiHandle != DRV_HANDLE_INVALID)
{
    /* Do further processing. */
}
else
{
```

```
    /* Call until the function returns a valid handle. */
}
```

Closing the Driver

Closes an opened-instance of the SQI driver

Example:

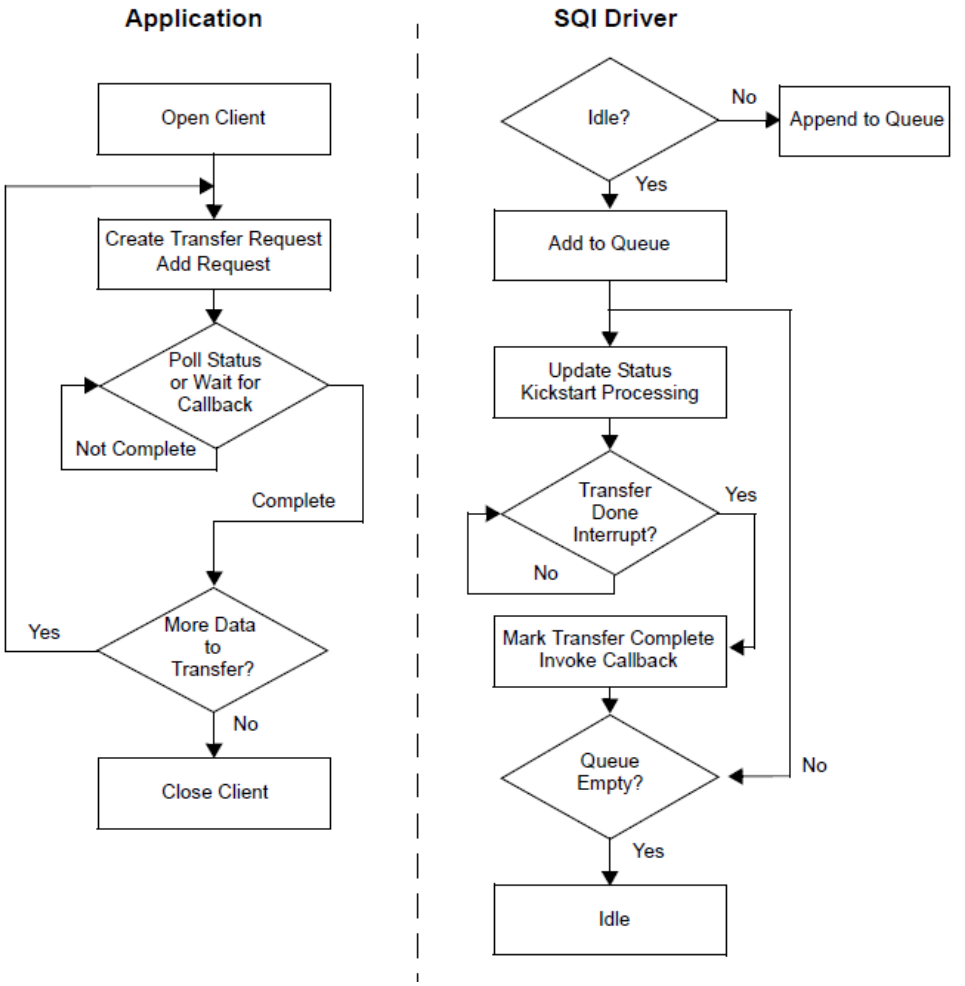
```
DRV_HANDLE handle; // Returned from DRV_SQI_Open
DRV_SQI_Close(handle);
```

Client Data Transfer Functions

Provides information on the client data transfer functions provided in the SQI Driver Library.

Description

Client data transfer functionality provides API interfaces for the data transfer operation. The following diagram illustrates the data transfer model.



Applications need to perform the following steps to transfer data using the SQI Driver:

1. The system should have completed necessary initialization and the DRV_SQI_Tasks should be running in an interrupt environment.
2. Open_the driver using DRV_SQI_Open with the necessary intent. The application should wait call the DRV_SQI_Open until the function returns a valid open handle.
3. Register callback function using the DRV_SQI_EventHandlerSet.
4. Add a transfer request using the buffer using the DRV_SQI_TransferData function. The reads or writes of blocks of data generally involves sending down the read or a write command, the address on the device from/to which data is to be read/written. The client also has to specify the source or destination buffer and the number of bytes to be read or written. The client builds an array of transfer elements containing this information and passes the array and the number of elements of the array as part of this transfer request.
5. Check for the current transfer status using DRV_SQI_CommandStatus until the transfer progress is DRV_SQI_COMMAND_COMPLETED, or wait for the callback to be called.
6. When the client has no more data to be transferred, the client can close the driver using DRV_SQI_Close.

Example:

```
#define READ_BUF_SIZE 512

uint8_t readBuffer[READ_BUF_SIZE] __attribute__((coherent, aligned(16)));
uint8_t command [5] __attribute__((coherent, aligned(16)) = {0x0B, 0x00, 0x00, 0x00, 0x0FF};
uint8_t numElements = 0;
DRV_SQI_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;
DRV_SQI_TRANSFER_ELEMENT xferData[2];

// mySQIHandle is the handle returned by the DRV_SQI_Open function.
// Setup the transfer elements.

xferData[0].data = &command[0];
xferData[0].length = sizeof(command);
xferData[0].flag = (DRV_SQI_FLAG_MODE_SINGLE_LANE);

xferData[1].data = readBuffer;
xferData[1].length = READ_BUF_SIZE;
xferData[1].flag = (DRV_SQI_FLAG_MODE_QUAD_LANE | DRV_SQI_FLAG_DIR_READ | DRV_SQI_FLAG_DEASSERT_CS);

DRV_SQI_TransferData(mySQIHandle, &commandHandle, 0, xferData, 2);

if(DRV_SQI_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Transfer operation queued successfully. Wait for the
    // completion event.
}

// Transfer completion can be tracked either by polling on the commandHandle or waiting
// for the event using the event callback function.
status = DRV_SQI_CommandStatus(mySQIHandle, commandHandle);
if(status == DRV_SQI_COMMAND_COMPLETED)
{
    // Operation Done
}

// Event handler.
void APP_SQIEventHandler
(
    DRV_SQI_EVENT event,
    DRV_SQI_COMMAND_HANDLE handle,
    uintptr_t context
)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event
    // handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SQI_EVENT_COMMAND_COMPLETE:
            // This means the operation was completed
            // successfully.
            break;

        case DRV_SQI_EVENT_COMMAND_ERROR:
            // Operation failed. Handle the error.
            break;

        default:
            break;
    }
}
```

Configuring the Library

Macros

	Name	Description
	DRV_SQI_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_SQI_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_SQI_DMA_BUFFER_DESCRIPTORS_NUMBER	Selects the maximum number of DMA Buffer descriptors to be used by the driver.
	DRV_SQI_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_SQI_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode

Description

The configuration of the SQI driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SQI driver. Based on the selections made, the SQI driver may support the selected features. These configuration settings will apply to all instances of the SQI driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_SQI_BUFFER_OBJECT_NUMBER Macro

Selects the maximum number of buffer objects

File

drv_sqi_config_template.h

C

```
#define DRV_SQI_BUFFER_OBJECT_NUMBER 5
```

Description

SQI Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The SQI Driver can queue up DRV_SQI_BUFFER_OBJECT_NUMBER of read/write/erase requests before return a DRV_SQI_BUFFER_HANDLE_INVALID due to the queue being full. Buffer objects are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SQI_CLIENTS_NUMBER Macro

Selects the maximum number of clients

File

drv_sqi_config_template.h

C

```
#define DRV_SQI_CLIENTS_NUMBER 1
```

Description

SQI maximum number of clients

This definition selects the maximum number of clients that the SQI driver can supported at run time. This constant defines the total number of SQI driver clients that will be available to all instances of the SQI driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SQI_DMA_BUFFER_DESCRIPTORS_NUMBER Macro

Selects the maximum number of DMA Buffer descriptors to be used by the driver.

File

drv_sqi_config_template.h

C

```
#define DRV_SQI_DMA_BUFFER_DESCRIPTORS_NUMBER 4
```

Description

SQI Driver maximum number DMA Buffer Descriptors

This definition selects the maximum number of DMA buffer descriptor objects. The SQI Driver can queue up to DRV_SQI_DMA_BUFFER_DESCRIPTORS_NUMBER of transactions to be processed by the hardware. DMA buffer desired are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SQI_INSTANCES_NUMBER Macro

Selects the maximum number of Driver instances that can be supported by the dynamic driver.

File

drv_sqi_config_template.h

C

```
#define DRV_SQI_INSTANCES_NUMBER 1
```

Description

SQI Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SQI_INTERRUPT_MODE Macro

Macro specifies operation of the driver to be in the interrupt mode or polled mode

File

drv_sqi_config_template.h

C

```
#define DRV_SQI_INTERRUPT_MODE true
```

Description

SQI interrupt and polled mode operation control

This macro specifies operation of the driver to be in the interrupt mode or polled mode

- true - Select if interrupt mode of SQI operation is desired
- false - Select if polling mode of SQI operation is desired

Not defining this option to true or false will result in build error.

Remarks

This macro is mandatory when building the driver for dynamic operation.

Building the Library

This section lists the files that are available in the SQI Driver Library.

Description

This section list the files that are available in the \src folder of the SQI Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/sqi`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_sqi.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_sqi.c	This file contains the source code for the dynamic implementation of the SQI driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SQI Driver Library depends on the following modules:

- Clock System Service Library

Optional Dependencies

- Interrupt System Service Library (used when task is running in Interrupt mode)

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_SQI_Initialize	Initializes the SQI instance for the specified driver index
	DRV_SQI_Deinitialize	Deinitializes the specified instance of the SQI driver module
	DRV_SQI_Status	Gets the current status of the SQI driver module.
	DRV_SQI_Tasks	Maintains the driver's task state machine.

b) Client Setup Functions

	Name	Description
	DRV_SQI_Open	Opens the specified SQI driver instance and returns a handle to it.
	DRV_SQI_Close	Closes an opened-instance of the SQI driver
	DRV_SQI_CommandStatus	Gets the current status of the transfer request.
	DRV_SQI_EventHandlerSet	Allows a client to register an event handling function, which the driver can invoke when the queued transfer request has completed.

c) Data Transfer Functions

	Name	Description
	DRV_SQI_TransferData	Queue a data transfer operation on the specified SQI device.
	DRV_SQI_TransferFrames	Queue a transfer request operation on the SQI device.

d) Data Types and Constants

	Name	Description
	DRV_SQI_COMMAND_HANDLE	Handle to identify the transfer request queued at the SQI driver.
	DRV_SQI_COMMAND_STATUS	Specifies the status of the transfer request.
	DRV_SQI_EVENT	Identifies the possible events that can result from a transfer request.
	DRV_SQI_EVENT_HANDLER	Pointer to a SQI Driver Event handler function
	DRV_SQI_SPI_OPERATION_MODE	Enumeration of the SPI mode of operation supported by the SQI Controller.
	DRV_SQI_TRANSFER_FLAGS	Enumeration of the configuration options associated with a single transfer element.
	DRV_SQI_TransferElement	Defines the data transfer element of the SQI driver.
	DRV_SQI_COMMAND_HANDLE_INVALID	Identifies an invalid command handle.
	DRV_SQI_INDEX_0	SQI driver index definitions.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE	This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS	Enables 32-bit addressing instead of 24-bit addressing.
	DRV_SQI_FLAG_ADDR_ENABLE	This is macro DRV_SQI_FLAG_ADDR_ENABLE.
	DRV_SQI_FLAG_ADDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_ADDR_ENABLE_MASK.
	DRV_SQI_FLAG_ADDR_ENABLE_POS	Address Enable Macro.
	DRV_SQI_FLAG_CRM_ENABLE	This is macro DRV_SQI_FLAG_CRM_ENABLE.
	DRV_SQI_FLAG_CRM_ENABLE_MASK	This is macro DRV_SQI_FLAG_CRM_ENABLE_MASK.
	DRV_SQI_FLAG_CRM_ENABLE_POS	Continuous Read Mode Enable Macro.
	DRV_SQI_FLAG_DATA_DIRECTION_MASK	This is macro DRV_SQI_FLAG_DATA_DIRECTION_MASK.
	DRV_SQI_FLAG_DATA_DIRECTION_POS	Macros to select the direction of the transfers.
	DRV_SQI_FLAG_DATA_DIRECTION_READ	This is macro DRV_SQI_FLAG_DATA_DIRECTION_READ.
	DRV_SQI_FLAG_DATA_DIRECTION_WRITE	This is macro DRV_SQI_FLAG_DATA_DIRECTION_WRITE.
	DRV_SQI_FLAG_DATA_ENABLE	This is macro DRV_SQI_FLAG_DATA_ENABLE.
	DRV_SQI_FLAG_DATA_ENABLE_MASK	This is macro DRV_SQI_FLAG_DATA_ENABLE_MASK.
	DRV_SQI_FLAG_DATA_ENABLE_POS	Data Enable Macro.
	DRV_SQI_FLAG_DATA_TARGET_MASK	This is macro DRV_SQI_FLAG_DATA_TARGET_MASK.
	DRV_SQI_FLAG_DATA_TARGET_MEMORY	This is macro DRV_SQI_FLAG_DATA_TARGET_MEMORY.
	DRV_SQI_FLAG_DATA_TARGET_POS	Macros to select the source and destination of a transfer.
	DRV_SQI_FLAG_DATA_TARGET_REGISTER	This is macro DRV_SQI_FLAG_DATA_TARGET_REGISTER.
	DRV_SQI_FLAG_DDR_ENABLE	This is macro DRV_SQI_FLAG_DDR_ENABLE.
	DRV_SQI_FLAG_DDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_DDR_ENABLE_MASK.
	DRV_SQI_FLAG_DDR_ENABLE_POS	DDR Enable Macro.
	DRV_SQI_FLAG_INSTR_ENABLE	This is macro DRV_SQI_FLAG_INSTR_ENABLE.
	DRV_SQI_FLAG_INSTR_ENABLE_MASK	This is macro DRV_SQI_FLAG_INSTR_ENABLE_MASK.
	DRV_SQI_FLAG_INSTR_ENABLE_POS	Macros listing the bitmap values for the flags member of the DRV_SQI_TransferFrame structure. Instruction Enable Macro.
	DRV_SQI_FLAG_OPT_ENABLE	This is macro DRV_SQI_FLAG_OPT_ENABLE.
	DRV_SQI_FLAG_OPT_ENABLE_MASK	This is macro DRV_SQI_FLAG_OPT_ENABLE_MASK.
	DRV_SQI_FLAG_OPT_ENABLE_POS	Option Enable Macro.
	DRV_SQI_FLAG_OPT_LENGTH	This is macro DRV_SQI_FLAG_OPT_LENGTH.
	DRV_SQI_FLAG_OPT_LENGTH_1BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_1BIT.
	DRV_SQI_FLAG_OPT_LENGTH_2BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_2BIT.
	DRV_SQI_FLAG_OPT_LENGTH_4BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_4BIT.
	DRV_SQI_FLAG_OPT_LENGTH_8BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_8BIT.
	DRV_SQI_FLAG_OPT_LENGTH_MASK	This is macro DRV_SQI_FLAG_OPT_LENGTH_MASK.
	DRV_SQI_FLAG_OPT_LENGTH_POS	Macros to enable and specify the option length.
	DRV_SQI_FLAG_SQI_CS_NUMBER	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER.
	DRV_SQI_FLAG_SQI_CS_NUMBER_0	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_0.
	DRV_SQI_FLAG_SQI_CS_NUMBER_1	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_1.
	DRV_SQI_FLAG_SQI_CS_NUMBER_2	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_2.
	DRV_SQI_FLAG_SQI_CS_NUMBER_3	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_3.
	DRV_SQI_FLAG_SQI_CS_NUMBER_MASK	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_MASK.

	DRV_SQI_FLAG_SQI_CS_NUMBER_POS	Macros to select the SQI CS Line Number to be used for the current transfer • frame.
	DRV_SQI_LANE_CONFIG	Defines the SQI lane configuration options.
	DRV_SQI_TransferFrame	Defines the transfer frame of the SQI driver.

Description

This section describes the API functions of the SQI Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_SQI_Initialize Function

Initializes the SQI instance for the specified driver index

File

drv_sqi.h

C

```
SYS_MODULE_OBJ DRV_SQI_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT *const init);
```

Returns

Returns a valid handle to a driver instance object on success. Otherwise returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the SQI driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other SQI routines are called.

This routine should only be called once during system initialization unless DRV_SQI_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to initialize, it will be reported by the DRV_SQI_Status operation. The system must use DRV_SQI_Status to find out when the driver is in the ready state.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SQI Driver.

SYS_MODULE_OBJ  objectHandle;
TODO:Replace with appropriate init snippet.
    // SQI Driver Initialization Data
    const DRV_SQI_INIT drvSqiInit =
    {
        .sqiId = SQI_ID_0,
        .interruptSource = INT_SOURCE_SQI1,
        .enabledDevices = DRV_SQI_ENABLE_BOTH_DEVICES,
        .clockDivider = DRV_SQI_CLK_DIV_1,
        .devCfg[0].spiMode = DRV_SQI_SPI_MODE_0,
        .devCfg[0].lsbFirst = true,
        .devCfg[1].spiMode = DRV_SQI_SPI_MODE_3,
        .devCfg[1].lsbFirst = false,
    };

    objectHandle = DRV_SQI_Initialize(DRV_SQI_INDEX_0, (SYS_MODULE_INIT*)&drvSqiInit);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized.
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SQI_Initialize
(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
);

DRV_SQI_Deinitialize Function

Deinitializes the specified instance of the SQI driver module

File

drv_sqi.h

C

```
void DRV_SQI_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SQI driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_SQI_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_SQI_Initialize routine

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SQI_Initialize
SYS_STATUS          status;

DRV_SQI_Deinitialize(object);

status = DRV_SQI_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later to know if the driver is deinitialized.
}
```

Function

void DRV_SQI_Deinitialize
(
SYS_MODULE_OBJ object
);

DRV_SQI_Status Function

Gets the current status of the SQI driver module.

File

drv_sqi.h

C

```
SYS_STATUS DRV_SQI_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and can accept transfer requests.

SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the SQI driver module.

Remarks

This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_SQI_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SQI_Initialize
SYS_STATUS          sqiStatus;

sqiStatus = DRV_SQI_Status(object);
else if (SYS_STATUS_ERROR >= sqiStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SQI_Initialize routine

Function

SYS_STATUS DRV_SQI_Status

(

SYS_MODULE_OBJ object

);

DRV_SQI_Tasks Function

Maintains the driver's task state machine.

File

drv_sqi.h

C

```
void DRV_SQI_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal task state machine.

Remarks

This routine may either be called by the system's task routine(SYS_Tasks) or the from the interrupt service routine of the peripheral.

Preconditions

The DRV_SQI_Initialize routine must have been called for the specified SQI driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SQI_Initialize

while (true)
{
    DRV_SQI_Tasks (object);
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SQI_Initialize)

Function

void DRV_SQI_Tasks

(

SYS_MODULE_OBJ object

);

b) Client Setup Functions

DRV_SQI_Open Function

Opens the specified SQI driver instance and returns a handle to it.

File

drv_sqi.h

C

```
DRV_HANDLE DRV_SQI_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SQI_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified SQI driver instance and provides a handle identifying the SQI driver instance. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_SQI_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has has already been opened, it cannot be opened exclusively.

Preconditions

Function DRV_SQI_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SQI_Open(DRV_SQI_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened.
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver.

Function

```
  DRV_HANDLE DRV_SQI_Open
(
const SYS_MODULE_INDEX index,
const    DRV_IO_INTENT ioIntent
);
```

DRV_SQI_Close Function

Closes an opened-instance of the SQI driver

File

drv_sqi.h

C

```
void DRV_SQI_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the SQI driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SQI_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SQI_Initialize routine must have been called for the specified SQI driver instance.

DRV_SQI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SQI_Open

DRV_SQI_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

```
void DRV_SQI_Close
(
const    DRV_HANDLE handle
);
```

DRV_SQI_CommandStatus Function

Gets the current status of the transfer request.

File

drv_sqi.h

C

```
DRV_SQI_COMMAND_STATUS DRV_SQI_CommandStatus(const DRV_HANDLE handle, const DRV_SQI_COMMAND_HANDLE
commandHandle);
```

Returns

A DRV_SQI_COMMAND_STATUS value describing the current status of the transfer request. Returns DRV_SQI_COMMAND_ERROR_UNKNOWN if the client handle or the handle is not valid.

Description

This routine gets the current status of the tranfer request. The application must use this routine where the status of a scheduled transfer request needs to polled on. The function may return DRV_SQI_COMMAND_COMPLETED in a case where the handle has expired. A handle expires when the internal buffer object is re-assigned to another transfer request. It is recommended that this function be called regularly in order to track the status of the transfer request correctly.

The application can alternatively register an event handler to receive the transfer completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status of the transfer request.

Preconditions

The DRV_SQI_Initialize() routine must have been called.

The DRV_SQI_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                  handle;         // Returned from DRV_SQI_Open
DRV_SQI_COMMAND_HANDLE      commandHandle;
DRV_SQI_COMMAND_STATUS      status;

status = DRV_SQI_CommandStatus(handle, commandHandle);
if(status == DRV_SQI_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_SQI_COMMAND_STATUS DRV_SQI_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SQI_COMMAND_HANDLE commandHandle

);

DRV_SQI_EventHandlerSet Function

Allows a client to register an event handling function, which the driver can invoke when the queued transfer request has completed.

File

drv_sqi.h

C

```
void DRV_SQI_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client queues a transfer request with the driver, it is provided with a handle identifying the transfer request that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any transfer operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SQI_Initialize() routine must have been called for the specified SQI driver instance.

The DRV_SQI_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

DRV_SQI_TransferFrame xferFrame;
DRV_SQI_COMMAND_HANDLE commandHandle;

// drvSQIHandle is the handle returned by the DRV_SQI_Open function.
// Client registers an event handler with driver. This is done once.
DRV_SQI_EventHandlerSet(drvSQIHandle, APP_SQIEventHandler, (uintptr_t)&myAppObj);

DRV_SQI_Read(drvSQIHandle, &commandHandle, &xferFrame, 1);

if(DRV_SQI_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event handler.
void APP_SQIEventHandler
(
    DRV_SQI_EVENT event,
    DRV_SQI_COMMAND_HANDLE handle,
    uintptr_t context
)
{
    // The context handle was set to an application specific object. It is
    // now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SQI_EVENT_COMMAND_COMPLETE:
            // This means the operation was completed successfully.
            break;

        case DRV_SQI_EVENT_COMMAND_ERROR:
            // Operation failed. Handle the error.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SQI_EventHandlerSet
(
const DRV_HANDLE handle,
const void *eventHandler,
const uintptr_t context
);

c) Data Transfer Functions

DRV_SQI_TransferData Function

Queue a data transfer operation on the specified SQI device.

File

drv_sqi.h

C

```
void DRV_SQI_TransferData(DRV_HANDLE handle, DRV_SQI_COMMAND_HANDLE * commandHandle, uint8_t sqiDevice,
DRV_SQI_TransferElement * xferData, uint8_t numElements);
```

Returns

The handle to the command request is returned in the commandHandle argument. It will be DRV_SQI_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine queues a data transfer operation on the specified SQI device. The reads or writes of blocks of data generally involves sending down the read or a write command, the address on the device from/to which data is to be read/written. The client also has to specify the source or destination buffer and the number of bytes to be read or written. The client builds an array of transfer elements containing these information and passes the array and the number of elements of the array as part of this transfer operation. If an event handler is registered with the driver the event handler would be invoked with the status of the operation once the operation has been completed. The function returns DRV_SQI_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the transfer element is NULL or number of transfer elements is zero
- if a buffer object could not be allocated to the request

Remarks

None.

Preconditions

The DRV_SQI_Initialize routine must have been called for the specified SQI driver instance.

DRV_SQI_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
#define READ_BUF_SIZE 512

uint8_t readBuffer[READ_BUF_SIZE] __attribute__((coherent, aligned(16)));
uint8_t command [5] __attribute__((coherent, aligned(16)) = {0x0B, 0x00, 0x00, 0x00, 0x0FF};
uint8_t numElements = 0;
DRV_SQI_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;
DRV_SQI_TransferElement xferData[2];

// mySQIHandle is the handle returned by the DRV_SQI_Open function.
// Setup the transfer elements.
```

```
xferData[0].data = &command[0];
xferData[0].length = sizeof(command);
xferData[0].flag = (DRV_SQI_FLAG_MODE_SINGLE_LANE);

xferData[1].data = readBuffer;
xferData[1].length = READ_BUF_SIZE;
xferData[1].flag = (DRV_SQI_FLAG_MODE_QUAD_LANE | DRV_SQI_FLAG_DIR_READ | DRV_SQI_FLAG_DEASSERT_CS);

DRV_SQI_TransferData(mySQIHandle, &commandHandle, 0, xferData, 2);

if(DRV_SQI_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Transfer operation queued successfully. Wait for the completion event.
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sqiDevice	The SQI device index on which the operation is to be performed.
xferData	Pointer to the transfer elements array.
numElements	Number of elements in the transfer elements array.

Function

void DRV_SQI_TransferData
(
    DRV_HANDLE handle,
    DRV_SQI_COMMAND_HANDLE *commandHandle,
uint8_t sqiDevice,
    DRV_SQI_TransferElement *xferData,
uint8_t numElements
);

DRV_SQI_TransferFrames Function

Queue a transfer request operation on the SQI device.

File

drv_sqi.h

C

```
void DRV_SQI_TransferFrames(DRV_HANDLE handle, DRV_SQI_COMMAND_HANDLE * commandHandle,
DRV_SQI_TransferFrame * frame, uint8_t numFrames);
```

Returns

The handle to the transfer request is returned in the commandHandle argument. It will be DRV_SQI_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine queues a transfer request operation on the SQI device. In order to perform any operation on the sqi flash device, a one byte instruction specifying the operation to be performed needs to be sent out. This is followed by optional address from/to which data is to be read/written, option, dummy and data bytes.

If an event handler is registered with the driver the event handler would be invoked with the status of the operation once the operation has been completed. The function returns DRV_SQI_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the transfer element is NULL or number of transfer elements is zero
- if a buffer object could not be allocated to the request

Remarks

None.

Preconditions

The DRV_SQI_Initialize routine must have been called for the specified SQI driver instance.

DRV_SQI_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
#define READ_BUF_SIZE 512

uint8_t readBuffer[READ_BUF_SIZE];
uint8_t numElements = 0;
DRV_SQI_COMMAND_HANDLE cmdHandle;
DRV_SQI_TransferFrame xferFrame;

DRV_SQI_TransferFrame *frame = &xferFrame;
frame->instruction = 0x6B;
frame->address = 0x00;
frame->data = readBuffer;
frame->length = READ_BUF_SIZE;
frame->laneCfg = DRV_SQI_LANE_QUAD_DATA;
frame->numDummyBytes = 8;
frame->flags = (DRV_SQI_FLAG_INSTR_ENABLE_MASK | DRV_SQI_FLAG_DATA_ENABLE_MASK |
    DRV_SQI_FLAG_ADDR_ENABLE_MASK | DRV_SQI_FLAG_DATA_TARGET_MEMORY |
    DRV_SQI_FLAG_DATA_DIRECTION_READ);
DRV_SQI_TransferFrames (sqiHandle, &cmdHandle, frame, 1);
if (cmdHandle == DRV_SQI_COMMAND_HANDLE_INVALID)
{
    // handle the failure.
}
else
{
    // continue with the rest of the operation.
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the handle to the track the status of the transfer request.
frame	Pointer to the transfer frame array.
numFrames	Number of elements in the transfer frame array.

Function

void DRV_SQI_TransferFrames

(

DRV_HANDLE handle,

DRV_SQI_COMMAND_HANDLE *commandHandle,

DRV_SQI_TransferFrame *frame,

uint8_t numFrames

);

d) Data Types and Constants

DRV_SQI_COMMAND_HANDLE Type

Handle to identify the transfer request queued at the SQI driver.

File

drv_sqi.h

C

```
typedef uintptr_t DRV_SQI_COMMAND_HANDLE;
```

Description

SQI Driver Command Handle

A command handle is returned by a call to the DRV_SQI_TransferFrames () function. This handle allows the application to track the completion of the request. This command handle is also returned to the client along with the event that has occurred with respect to the request. This allows the application to connect the event to a specific transfer request in case where multiple requests are queued.

The command handle associated with the transfer request expires when the client has been notified of the completion of the request (after event handler function that notifies the client returns) or after the request has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SQI_COMMAND_STATUS Enumeration

Specifies the status of the transfer request.

File

drv_sqi.h

C

```
typedef enum {
  DRV_SQI_COMMAND_COMPLETED,
  DRV_SQI_COMMAND_QUEUED,
  DRV_SQI_COMMAND_IN_PROGRESS,
  DRV_SQI_COMMAND_ERROR_UNKNOWN
} DRV_SQI_COMMAND_STATUS;
```

Members

Members	Description
DRV_SQI_COMMAND_COMPLETED	Command completed.
DRV_SQI_COMMAND_QUEUED	Command is pending.
DRV_SQI_COMMAND_IN_PROGRESS	Command is being processed
DRV_SQI_COMMAND_ERROR_UNKNOWN	There was an error while processing the command.

Description

SQI Driver Command Status

This enumeration identifies the possible status values associated with a transfer request. The client can retrieve the status by calling the DRV_SQI_CommandStatus () function and passing the command handle associated with the request.

Remarks

None.

DRV_SQI_EVENT Enumeration

Identifies the possible events that can result from a transfer request.

File

drv_sqi.h

C

```
typedef enum {
  DRV_SQI_EVENT_COMMAND_COMPLETE = 0,
  DRV_SQI_EVENT_COMMAND_ERROR
} DRV_SQI_EVENT;
```

Members

Members	Description
DRV_SQI_EVENT_COMMAND_COMPLETE = 0	Operation has been completed successfully.
DRV_SQI_EVENT_COMMAND_ERROR	There was an error during the operation

Description

SQI Driver Events

This enumeration identifies the possible events that can result from a transfer request issued by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SQI_EventHandlerSet function when a request is completed.

DRV_SQI_EVENT_HANDLER Type

Pointer to a SQI Driver Event handler function

File

drv_sqi.h

C

```
typedef void (* DRV_SQI_EVENT_HANDLER)(DRV_SQI_EVENT event, DRV_SQI_COMMAND_HANDLE commandHandle, void
*context);
```

Returns

None.

Description

SQI Driver Event Handler Function Pointer data type.

This data type defines the required function signature for the SQI driver event handling callback function. A client must register a pointer to a event handling function the signature(parameter and return value types) of which should match the types specified by this function pointer in order to receive transfer request related event call backs from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SQI_EVENT_COMMAND_COMPLETE, that the operation associated with the transfer request was completed successfully. If the event is DRV_SQI_EVENT_COMMAND_ERROR, there was an error while executing the transfer request.

The context parameter contains context details provided by the client as part of registering the event handler function. This context value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) of the client that made the request.

Example

```
void MyAppCommandEventHandler
(
    DRV_SQI_EVENT event,
    DRV_SQI_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT)context;

    switch(event)
    {
        case DRV_SQI_EVENT_COMMAND_COMPLETE:
            // Handle the completed transfer request.
            break;

        case DRV_SQI_EVENT_COMMAND_ERROR:
        default:
            // Handle the failed transfer request.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle identifying the transfer request to which this event relates

context	Value identifying the context of the application that registered the event handling function.

DRV_SQI_SPI_OPERATION_MODE Enumeration

Enumeration of the SPI mode of operation supported by the SQI Controller.

File

drv_sqi.h

C

```
typedef enum {
  DRV_SQI_SPI_MODE_0 = 0,
  DRV_SQI_SPI_MODE_3 = 3
} DRV_SQI_SPI_OPERATION_MODE;
```

Members

Members	Description
DRV_SQI_SPI_MODE_0 = 0	CPOL = 0 and CPHA = 0. SCK Idle state = LOW
DRV_SQI_SPI_MODE_3 = 3	CPOL = 1 and CPHA = 1. SCK Idle state = HIGH

Description

SQI SPI Mode of operation

This enumeration lists the SPI mode of operation supported by the SQI controller. In MODE 0 of operation: CPOL = 0 and CPHA = 0. SCK Idle state = LOW

In MODE 3 of operation: CPOL = 1 and CPHA = 1. SCK Idle state = HIGH

In both MODE 0 and MODE 3 of operation the: SQI Data Input is sampled on the rising edge of the SQI Clock SQI Data is Output on the falling edge of the SQI Clock

Remarks

None

DRV_SQI_TRANSFER_FLAGS Enumeration

Enumeration of the configuration options associated with a single transfer element.

File

drv_sqi.h

C

```
typedef enum {
  DRV_SQI_FLAG_MODE_SINGLE_LANE = 0x00,
  DRV_SQI_FLAG_MODE_DUAL_LANE = 0x01,
  DRV_SQI_FLAG_MODE_QUAD_LANE = 0x02,
  DRV_SQI_FLAG_DDR_MODE = 0x04,
  DRV_SQI_FLAG_DEASSERT_CS = 0x08,
  DRV_SQI_FLAG_DIR_READ = 0x80
} DRV_SQI_TRANSFER_FLAGS;
```

Members

Members	Description
DRV_SQI_FLAG_MODE_SINGLE_LANE = 0x00	Bits 0-1: Indicates the Lane configuration to be used.
DRV_SQI_FLAG_DDR_MODE = 0x04	Bit 2: This bit indicates if DDR or SDR mode of operation is to be used.
DRV_SQI_FLAG_DEASSERT_CS = 0x08	Bit 3: This bit indicates if CS is to be de-asserted at the end of this • transaction.
DRV_SQI_FLAG_DIR_READ = 0x80	Bit 7: This bit indicates if the operation is a read or a write.

Description

Flags associated with the SQI Driver Transfer element.

This enumeration lists the various configuration options associated with a single transfer element(Refer to the data structure DRV_SQI_TransferElement). The client can specify one or more of these as configuration parameters of a single transfer element.

Remarks

None

DRV_SQI_TransferElement Structure

Defines the data transfer element of the SQI driver.

File

drv_sqi.h

C

```
typedef struct {
  uint8_t * data;
  uint32_t length;
  uint8_t flag;
} DRV_SQI_TransferElement;
```

Members

Members	Description
uint8_t * data;	Pointer to the source or destination buffer
uint32_t length;	Length of the buffer in bytes.
uint8_t flag;	This is a bitmap used to indicate the configuration options to be used • for this transfer element. One or more values of the enumeration • DRV_SQI_TRANSFER_FLAGS can be passed as part of this flag.

Description

SQI Driver data transfer element.

This data type defines the composition of a single transfer element. A single element will consist of the pointer to the source of destination buffer, length of the data to be transferred or received and the various configuration options to be used for the element. The configuration options also indicate if data is transferred to/from the device. A client builds an array of such transfer elements and passes the array and the number of elements of the array as part of the read or write operation.

Remarks

None.

DRV_SQI_COMMAND_HANDLE_INVALID Macro

Identifies an invalid command handle.

File

drv_sqi.h

C

```
#define DRV_SQI_COMMAND_HANDLE_INVALID ((DRV_SQI_COMMAND_HANDLE)(-1))
```

Description

SQI Driver Invalid Command Handle

This is the definition of an invalid command handle. An invalid command handle is returned by DRV_SQI_TransferFrames() function if the transfer request was not queued.

Remarks

None.

DRV_SQI_INDEX_0 Macro

SQI driver index definitions.

File

drv_sqi.h

C

```
#define DRV_SQI_INDEX_0 0
```

Description

SQI Driver Module Index Numbers

This constant provides the SQI driver index definition.

Remarks

This constant should be used in place of hard-coded numeric literal.

This value should be passed into the DRV_SQI_Initialize and DRV_SQI_Open functions to identify the driver instance in use.

DRV_SQI_FLAG_32_BIT_ADDR_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_32_BIT_ADDR_ENABLE(value) (DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE.

DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK (0x1U << DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK.

DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS (6)
```

Description

Enables 32-bit addressing instead of 24-bit addressing.

DRV_SQI_FLAG_ADDR_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_ADDR_ENABLE(value) (DRV_SQI_FLAG_ADDR_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_ADDR_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_ADDR_ENABLE.

DRV_SQI_FLAG_ADDR_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_ADDR_ENABLE_MASK (0x1U << DRV_SQI_FLAG_ADDR_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_ADDR_ENABLE_MASK.

DRV_SQI_FLAG_ADDR_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_ADDR_ENABLE_POS (1)
```

Description

Address Enable Macro.

DRV_SQI_FLAG_CRM_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_CRM_ENABLE(value) (DRV_SQI_FLAG_CRM_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_CRM_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_CRM_ENABLE.

DRV_SQI_FLAG_CRM_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_CRM_ENABLE_MASK (0x1U << DRV_SQI_FLAG_CRM_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_CRM_ENABLE_MASK.

DRV_SQI_FLAG_CRM_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_CRM_ENABLE_POS (5)
```

Description

Continuous Read Mode Enable Macro.

DRV_SQI_FLAG_DATA_DIRECTION_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_DIRECTION_MASK (0x1U << DRV_SQI_FLAG_DATA_DIRECTION_POS)
```

Description

This is macro DRV_SQI_FLAG_DATA_DIRECTION_MASK.

DRV_SQI_FLAG_DATA_DIRECTION_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_DIRECTION_POS (11)
```

Description

Macros to select the direction of the transfers.

DRV_SQI_FLAG_DATA_DIRECTION_READ Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_DIRECTION_READ (DRV_SQI_FLAG_DATA_DIRECTION_MASK & ((0x1U) <<
DRV_SQI_FLAG_DATA_DIRECTION_POS))
```

Description

This is macro DRV_SQI_FLAG_DATA_DIRECTION_READ.

DRV_SQI_FLAG_DATA_DIRECTION_WRITE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_DIRECTION_WRITE (DRV_SQI_FLAG_DATA_DIRECTION_MASK & ((0x0U) <<
DRV_SQI_FLAG_DATA_DIRECTION_POS))
```

Description

This is macro DRV_SQI_FLAG_DATA_DIRECTION_WRITE.

DRV_SQI_FLAG_DATA_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_ENABLE(value) (DRV_SQI_FLAG_DATA_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_DATA_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_DATA_ENABLE.

DRV_SQI_FLAG_DATA_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_ENABLE_MASK (0x1U << DRV_SQI_FLAG_DATA_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_DATA_ENABLE_MASK.

DRV_SQI_FLAG_DATA_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_ENABLE_POS (3)
```

Description

Data Enable Macro.

DRV_SQI_FLAG_DATA_TARGET_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_TARGET_MASK (0x1U << DRV_SQI_FLAG_DATA_TARGET_POS)
```

Description

This is macro DRV_SQI_FLAG_DATA_TARGET_MASK.

DRV_SQI_FLAG_DATA_TARGET_MEMORY Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_TARGET_MEMORY (DRV_SQI_FLAG_DATA_TARGET_MASK & ((0x1U) <<
DRV_SQI_FLAG_DATA_TARGET_POS))
```

Description

This is macro DRV_SQI_FLAG_DATA_TARGET_MEMORY.

DRV_SQI_FLAG_DATA_TARGET_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_TARGET_POS (10)
```

Description

Macros to select the source and destination of a transfer.

DRV_SQI_FLAG_DATA_TARGET_REGISTER Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DATA_TARGET_REGISTER (DRV_SQI_FLAG_DATA_TARGET_MASK & ((0x0U) <<
DRV_SQI_FLAG_DATA_TARGET_POS))
```

Description

This is macro DRV_SQI_FLAG_DATA_TARGET_REGISTER.

DRV_SQI_FLAG_DDR_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DDR_ENABLE(value) (DRV_SQI_FLAG_DDR_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_DDR_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_DDR_ENABLE.

DRV_SQI_FLAG_DDR_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DDR_ENABLE_MASK (0x1U << DRV_SQI_FLAG_DDR_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_DDR_ENABLE_MASK.

DRV_SQI_FLAG_DDR_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_DDR_ENABLE_POS (4)
```

Description

DDR Enable Macro.

DRV_SQI_FLAG_INSTR_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_INSTR_ENABLE(value) (DRV_SQI_FLAG_INSTR_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_INSTR_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_INSTR_ENABLE.

DRV_SQI_FLAG_INSTR_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_INSTR_ENABLE_MASK (0x1U << DRV_SQI_FLAG_INSTR_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_INSTR_ENABLE_MASK.

DRV_SQI_FLAG_INSTR_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_INSTR_ENABLE_POS (0)
```

Description

Macros listing the bitmap values for the flags member of the DRV_SQI_TransferFrame structure. Instruction Enable Macro.

DRV_SQI_FLAG_OPT_ENABLE Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_ENABLE(value) (DRV_SQI_FLAG_OPT_ENABLE_MASK & ((value) <<
DRV_SQI_FLAG_OPT_ENABLE_POS))
```

Description

This is macro DRV_SQI_FLAG_OPT_ENABLE.

DRV_SQI_FLAG_OPT_ENABLE_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_ENABLE_MASK (0x1U << DRV_SQI_FLAG_OPT_ENABLE_POS)
```

Description

This is macro DRV_SQI_FLAG_OPT_ENABLE_MASK.

DRV_SQI_FLAG_OPT_ENABLE_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_ENABLE_POS (2)
```

Description

Option Enable Macro.

DRV_SQI_FLAG_OPT_LENGTH Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH(value) (DRV_SQI_FLAG_OPT_LENGTH_MASK & ((value) <<
DRV_SQI_FLAG_OPT_LENGTH_POS))
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH.

DRV_SQI_FLAG_OPT_LENGTH_1BIT Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_1BIT (0x0U)
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH_1BIT.

DRV_SQI_FLAG_OPT_LENGTH_2BIT Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_2BIT (0x1U)
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH_2BIT.

DRV_SQI_FLAG_OPT_LENGTH_4BIT Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_4BIT (0x2U)
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH_4BIT.

DRV_SQI_FLAG_OPT_LENGTH_8BIT Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_8BIT (0x3U)
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH_8BIT.

DRV_SQI_FLAG_OPT_LENGTH_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_MASK (0x3U << DRV_SQI_FLAG_OPT_LENGTH_POS)
```

Description

This is macro DRV_SQI_FLAG_OPT_LENGTH_MASK.

DRV_SQI_FLAG_OPT_LENGTH_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_OPT_LENGTH_POS (8)
```

Description

Macros to enable and specify the option length.

DRV_SQI_FLAG_SQI_CS_NUMBER Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER(value) (DRV_SQI_FLAG_SQI_CS_NUMBER_MASK & ((value) <<
DRV_SQI_FLAG_SQI_CS_NUMBER_POS))
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER.

DRV_SQI_FLAG_SQI_CS_NUMBER_0 Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_0 (0x0U)
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_0.

DRV_SQI_FLAG_SQI_CS_NUMBER_1 Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_1 (0x1U)
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_1.

DRV_SQI_FLAG_SQI_CS_NUMBER_2 Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_2 (0x2U)
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_2.

DRV_SQI_FLAG_SQI_CS_NUMBER_3 Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_3 (0x3U)
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_3.

DRV_SQI_FLAG_SQI_CS_NUMBER_MASK Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_MASK (0x3U << DRV_SQI_FLAG_SQI_CS_NUMBER_POS)
```

Description

This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_MASK.

DRV_SQI_FLAG_SQI_CS_NUMBER_POS Macro

File

drv_sqi.h

C

```
#define DRV_SQI_FLAG_SQI_CS_NUMBER_POS (16)
```

Description

Macros to select the SQI CS Line Number to be used for the current transfer

- frame.

DRV_SQI_LANE_CONFIG Enumeration

Defines the SQI lane configuration options.

File

drv_sqi.h

C

```
typedef enum {
  DRV_SQI_LANE_SINGLE = 0,
  DRV_SQI_LANE_DUAL_DATA,
  DRV_SQI_LANE_QUAD_DATA,
  DRV_SQI_LANE_DUAL_ADDR_DATA,
  DRV_SQI_LANE_QUAD_ADDR_DATA,
```

```
  DRV_SQI_LANE_DUAL_ALL,
  DRV_SQI_LANE_QUAD_ALL
} DRV_SQI_LANE_CONFIG;
```

Members

Members	Description
DRV_SQI_LANE_SINGLE = 0	Instruction opcode, Address and Data are all sent in single lane
DRV_SQI_LANE_DUAL_DATA	Instruction opcode and Address are sent in single lane, while data is • sent using dual lane.
DRV_SQI_LANE_QUAD_DATA	Instruction opcode and Address are sent in single lane, while data is • sent using quad lane.
DRV_SQI_LANE_DUAL_ADDR_DATA	Instruction opcode is sent in single lane, Address and Data are sent • using dual lane.
DRV_SQI_LANE_QUAD_ADDR_DATA	Instruction opcode is sent in single lane, Address and Data are sent • using quad lane.
DRV_SQI_LANE_DUAL_ALL	Instruction opcode, Address and Data are sent using dual lanes.
DRV_SQI_LANE_QUAD_ALL	Instruction opcode, Address and Data are sent using quad lanes.

Description

SQI lane configuration options.

This enumeration lists the various lane configuration options provided by the driver.

Remarks

None.

DRV_SQI_TransferFrame Structure

Defines the transfer frame of the SQI driver.

File

drv_sqi.h

C

```
typedef struct {
  uint8_t instruction;
  uint32_t address;
  uint8_t * data;
  uint32_t length;
  DRV_SQI_LANE_CONFIG laneCfg;
  uint8_t option;
  uint8_t numDummyBytes;
  uint32_t flags;
} DRV_SQI_TransferFrame;
```

Members

Members	Description
uint8_t instruction;	8-bit instruction opcode.
uint32_t address;	24/32-bit address.
uint8_t * data;	Pointer to the source or destination buffer
uint32_t length;	Length of the buffer in bytes.
DRV_SQI_LANE_CONFIG laneCfg;	Lane Configuration.
uint8_t option;	Option code associated with the current command.
uint8_t numDummyBytes;	Optional number of dummy bytes associated with the current command.
uint32_t flags;	This is bit-map field providing various configuration options for the • current frame.

Description

SQI Driver transfer frame.

This data type defines the composition of a single transfer frame. In order to perform any operation on the SQI flash device, a one byte instruction specifying the operation to be performed needs to be sent out. This is followed by optional address from/to which data is to be read/written, option, dummy and data bytes.

The configuration options also indicate if data is transferred to/from the device.

Remarks

None.

Files

Files

Name	Description
drv_sqi.h	SQI Driver Interface Definition
drv_sqi_config_template.h	SQI driver configuration definitions.

Description

This section lists the source and header files used by the SQI Driver Library.

drv_sqi.h

SQI Driver Interface Definition

Enumerations

	Name	Description
	DRV_SQI_COMMAND_STATUS	Specifies the status of the transfer request.
	DRV_SQI_EVENT	Identifies the possible events that can result from a transfer request.
	DRV_SQI_LANE_CONFIG	Defines the SQI lane configuration options.
	DRV_SQI_SPI_OPERATION_MODE	Enumeration of the SPI mode of operation supported by the SQI Controller.
	DRV_SQI_TRANSFER_FLAGS	Enumeration of the configuration options associated with a single transfer element.

Functions

	Name	Description
	DRV_SQI_Close	Closes an opened-instance of the SQI driver
	DRV_SQI_CommandStatus	Gets the current status of the transfer request.
	DRV_SQI_Deinitialize	Deinitializes the specified instance of the SQI driver module
	DRV_SQI_EventHandlerSet	Allows a client to register an event handling function, which the driver can invoke when the queued transfer request has completed.
	DRV_SQI_Initialize	Initializes the SQI instance for the specified driver index
	DRV_SQI_Open	Opens the specified SQI driver instance and returns a handle to it.
	DRV_SQI_Status	Gets the current status of the SQI driver module.
	DRV_SQI_Tasks	Maintains the driver's task state machine.
	DRV_SQI_TransferData	Queue a data transfer operation on the specified SQI device.
	DRV_SQI_TransferFrames	Queue a transfer request operation on the SQI device.

Macros

	Name	Description
	DRV_SQI_COMMAND_HANDLE_INVALID	Identifies an invalid command handle.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE	This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_MASK.
	DRV_SQI_FLAG_32_BIT_ADDR_ENABLE_POS	Enables 32-bit addressing instead of 24-bit addressing.
	DRV_SQI_FLAG_ADDR_ENABLE	This is macro DRV_SQI_FLAG_ADDR_ENABLE.
	DRV_SQI_FLAG_ADDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_ADDR_ENABLE_MASK.
	DRV_SQI_FLAG_ADDR_ENABLE_POS	Address Enable Macro.
	DRV_SQI_FLAG_CRM_ENABLE	This is macro DRV_SQI_FLAG_CRM_ENABLE.
	DRV_SQI_FLAG_CRM_ENABLE_MASK	This is macro DRV_SQI_FLAG_CRM_ENABLE_MASK.
	DRV_SQI_FLAG_CRM_ENABLE_POS	Continuous Read Mode Enable Macro.
	DRV_SQI_FLAG_DATA_DIRECTION_MASK	This is macro DRV_SQI_FLAG_DATA_DIRECTION_MASK.
	DRV_SQI_FLAG_DATA_DIRECTION_POS	Macros to select the direction of the transfers.
	DRV_SQI_FLAG_DATA_DIRECTION_READ	This is macro DRV_SQI_FLAG_DATA_DIRECTION_READ.

	DRV_SQI_FLAG_DATA_DIRECTION_WRITE	This is macro DRV_SQI_FLAG_DATA_DIRECTION_WRITE.
	DRV_SQI_FLAG_DATA_ENABLE	This is macro DRV_SQI_FLAG_DATA_ENABLE.
	DRV_SQI_FLAG_DATA_ENABLE_MASK	This is macro DRV_SQI_FLAG_DATA_ENABLE_MASK.
	DRV_SQI_FLAG_DATA_ENABLE_POS	Data Enable Macro.
	DRV_SQI_FLAG_DATA_TARGET_MASK	This is macro DRV_SQI_FLAG_DATA_TARGET_MASK.
	DRV_SQI_FLAG_DATA_TARGET_MEMORY	This is macro DRV_SQI_FLAG_DATA_TARGET_MEMORY.
	DRV_SQI_FLAG_DATA_TARGET_POS	Macros to select the source and destination of a transfer.
	DRV_SQI_FLAG_DATA_TARGET_REGISTER	This is macro DRV_SQI_FLAG_DATA_TARGET_REGISTER.
	DRV_SQI_FLAG_DDR_ENABLE	This is macro DRV_SQI_FLAG_DDR_ENABLE.
	DRV_SQI_FLAG_DDR_ENABLE_MASK	This is macro DRV_SQI_FLAG_DDR_ENABLE_MASK.
	DRV_SQI_FLAG_DDR_ENABLE_POS	DDR Enable Macro.
	DRV_SQI_FLAG_INSTR_ENABLE	This is macro DRV_SQI_FLAG_INSTR_ENABLE.
	DRV_SQI_FLAG_INSTR_ENABLE_MASK	This is macro DRV_SQI_FLAG_INSTR_ENABLE_MASK.
	DRV_SQI_FLAG_INSTR_ENABLE_POS	Macros listing the bitmap values for the flags member of the DRV_SQI_TransferFrame structure. Instruction Enable Macro.
	DRV_SQI_FLAG_OPT_ENABLE	This is macro DRV_SQI_FLAG_OPT_ENABLE.
	DRV_SQI_FLAG_OPT_ENABLE_MASK	This is macro DRV_SQI_FLAG_OPT_ENABLE_MASK.
	DRV_SQI_FLAG_OPT_ENABLE_POS	Option Enable Macro.
	DRV_SQI_FLAG_OPT_LENGTH	This is macro DRV_SQI_FLAG_OPT_LENGTH.
	DRV_SQI_FLAG_OPT_LENGTH_1BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_1BIT.
	DRV_SQI_FLAG_OPT_LENGTH_2BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_2BIT.
	DRV_SQI_FLAG_OPT_LENGTH_4BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_4BIT.
	DRV_SQI_FLAG_OPT_LENGTH_8BIT	This is macro DRV_SQI_FLAG_OPT_LENGTH_8BIT.
	DRV_SQI_FLAG_OPT_LENGTH_MASK	This is macro DRV_SQI_FLAG_OPT_LENGTH_MASK.
	DRV_SQI_FLAG_OPT_LENGTH_POS	Macros to enable and specify the option length.
	DRV_SQI_FLAG_SQI_CS_NUMBER	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER.
	DRV_SQI_FLAG_SQI_CS_NUMBER_0	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_0.
	DRV_SQI_FLAG_SQI_CS_NUMBER_1	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_1.
	DRV_SQI_FLAG_SQI_CS_NUMBER_2	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_2.
	DRV_SQI_FLAG_SQI_CS_NUMBER_3	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_3.
	DRV_SQI_FLAG_SQI_CS_NUMBER_MASK	This is macro DRV_SQI_FLAG_SQI_CS_NUMBER_MASK.
	DRV_SQI_FLAG_SQI_CS_NUMBER_POS	Macros to select the SQI CS Line Number to be used for the current transfer • frame.
	DRV_SQI_INDEX_0	SQI driver index definitions.

Structures

	Name	Description
	DRV_SQI_TransferElement	Defines the data transfer element of the SQI driver.
	DRV_SQI_TransferFrame	Defines the transfer frame of the SQI driver.

Types

	Name	Description
	DRV_SQI_COMMAND_HANDLE	Handle to identify the transfer request queued at the SQI driver.
	DRV_SQI_EVENT_HANDLER	Pointer to a SQI Driver Event handler function

Description

SQI Driver Interface Definition

The SQI driver provides data structures and interfaces to manage the SQI controller. This file contains the data structures and interface definitions of the SQI driver.

File Name

drv_sqi.h

Company

Microchip Technology Inc.

drv_sqi_config_template.h

SQI driver configuration definitions.

Macros

	Name	Description
	DRV_SQI_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_SQI_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_SQI_DMA_BUFFER_DESCRIPTORS_NUMBER	Selects the maximum number of DMA Buffer descriptors to be used by the driver.
	DRV_SQI_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_SQI_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode

Description

SQI Driver Configuration Template Header file.

This template file describes all the mandatory and optional configuration macros that are needed for building the SQI driver. Do not include this file in source code.

File Name

drv_sqi_config_template.h

Company

Microchip Technology Inc.

SQI Flash Driver Library

This section describes the Serial Quad Interface (SQI) Flash Driver Library.

Introduction

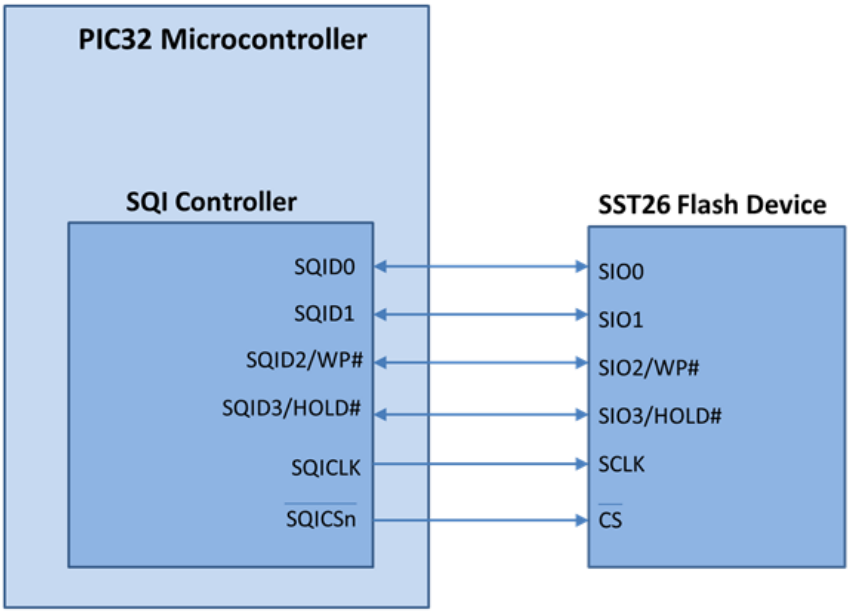
This library provides an interface to manage the SST26VF family of SQI Flash devices in different modes of operation.

Description

The MPLAB Harmony SST26 SQI Flash Driver provides a high-level interface to manage the SST26VF family of Flash devices over the SQI interface. The driver includes the following features:

- Provides application ready routines to perform block operations on the SQI Flash devices
- Supports Single, Dual, and Quad Lane modes
- Supports multi-client operation
- Provides data transfer events
- Supports non-blocking mode of operation only
- Thread-safe functions for use in RTOS applications

The SST26 Flash Driver uses the SQI module to establish the communication between SST26 Flash devices and Microchip microcontrollers. The following diagram shows the pin connections that are required to make the driver operational:



The SST26 Flash driver supports multi-client operation. This feature allows multiple application clients to access the same Flash device. Multiple instances of the driver can be used when multiple Flash devices are required to be part of the system.

Using the Library

This topic describes the basic architecture of the SQI Flash Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_sst26.h`

The interface to the SQI Flash Driver Library is defined in the header file. Any C language source (`.c`) file that uses the SQI Flash Driver library should include this header.

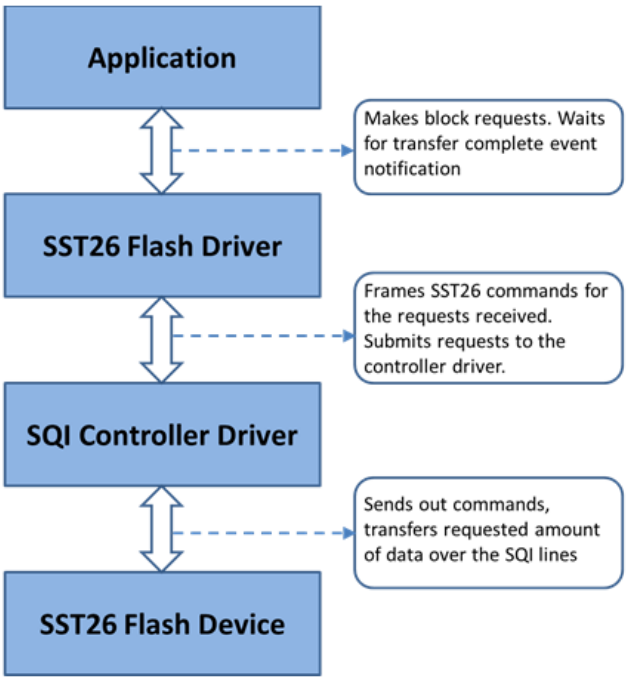
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the SQI Flash Driver Library with a convenient C language interface. This topic describes how that abstraction is modeled in software.

Description

The SST26 SQI Flash needs a specific set of commands to be given on its SQI interface along with the required address and data to do different operations. This driver abstracts these requirements and provide simple APIs that can be used to perform Erase, Write, and Read operations. The SQI Driver is used for this purpose. The following layered diagram depicts the communication between different modules.



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SQI Flash Driver module.

Library Interface Section	Description
System Functions	These functions are accessed by the MPLAB Harmony System module and allow the driver to be initialized, deinitialized, and maintained.
Core Client Functions	These functions allow the application client to open and close the driver.
Block Operation Functions	These functions enable the Flash module to be erased, written, and read (to/from).
Media Interface Functions	These functions provide media status and the Flash geometry.

How the Library Works

This topic provides information on how the SQI Flash Driver Library works.

Description

System Functions

SST26 Driver Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization each instance of the SST26 Flash driver would be initialized with the following configuration settings passed dynamically at run time using DRV_SST26_INIT, that are supported by the specific SST26 Flash driver:

- sqiDevice: The SQI controller supports a maximum of two slave devices. This identifies the SQI device index on which the flash device is located.

The DRV_SST26_Initialize function configures and initializes the SST26 Flash driver using the configuration information provided. It returns an object handle of the type SYS_MODULE_OBJ. This object handle would be used by other system interfaces such as DRV_SST26_Status, DRV_SST26_Tasks and DRV_SST26_Deinitialize.

Example:

```
/*** SST26 FLASH Driver Initialization Data ***/
const DRV_SST26_INIT drvSst26InitData0 =
{
    .sqiDevice = 1,
};
```

```
/* Initialize the SST26 Driver */
sysObj.drvSst26Obj0 = DRV_SST26_Initialize(DRV_SST26_INDEX_0,
                                           (SYS_MODULE_INIT *)&drvSst26InitData0);
```

SST26 Flash Driver Task Routine

The SST26 Driver task routine DRV_SST26_Tasks, will be called from the system task routine, SYS_Tasks. The driver task routine is responsible maintaining the driver state machine. The block operation requests from the application or from other modules are added to the driver queue. The task routine processes these queued requests by invoking the SQI driver routines for handling the transfer to the flash media.

SST26 Flash Driver Status

DRV_SST26_Status returns the current status of the SST26 Flash driver and is called by MPLAB Harmony. The application may not find the need to call this function directly.

Example:

```
SYS_MODULE_OBJ object;
// Returned from DRV_SST26_Initialize
SYS_STATUS sst26Status;

sst26Status = DRV_SST26_Status(object);
if (SYS_STATUS_ERROR >= sst26Status)
{
    // Handle error
}
```

Client Core Functions

Opening the Driver

For the application to start using an instance of the module, it must call the DRV_SST26_Open function repeatedly until a valid handle is returned by the driver. The application client uses this driver handle to access the driver functions.

For the various options available for I/O INTENT please refer to Data Types and Constants in the Library Interface section.

Example:

```
handle = DRV_SST26_Open(DRV_SST26_INDEX_0,
         DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    /* Call until the function returns a valid handle. */
}
else
{
    /* Do further processing. */
}
```

Closing the Driver

DRV_SST26_Close closes an opened-instance of the SST26 Flash driver, which also invalidates the driver handle. The application must open the driver again to obtain a valid handle.

Example:

```
DRV_HANDLE handle; // Returned from DRV_SST26_Open
DRV_SST26_Close(handle);
```

Client Block Operation Functions

The driver provides client interfaces to perform operations in terms of blocks. A block is a unit that represents the minimum amount of data that can be erased, written, or read. The block sizes may differ for Erase, Write, and Read operations. The DRV_SST26_GeometryGet function can be used to read out the geometry of the flash device. The geometry indicates the number of read, write and erase regions, blocks per region and the size of each block.

The DRV_SST26_Erase, DRV_SST26_Write, and DRV_SST26_Read functions are used to erase, write, and read the data to/from SST26 Flash devices. In addition to these functions, the driver also provides the DRV_SST26_EraseWrite function that combines the step of erasing a sector and then writing a page. The application can use this function if it wants to avoid having to explicitly delete a sector in order to update the pages contained in the sector.

These functions are non-blocking in nature and queue the operation request into the driver queue. All of the requests in the queue are executed by the DRV_SST26_Tasks function one-by-one. A command handle associated with the operation request is returned to the application client when the operation request is queued at the driver. This handle allows the application client to track the request as it progresses through the queue. The handle expires when the request processing is complete. The driver provides events (DRV_SST26_EVENT) that indicate the completion of the requests.

The following steps can be used for a simple Block Data Operation:

1. The system should have completed necessary initialization of the SQI Driver and the SST26 Flash Driver, and the DRV_SST26_Tasks function should be running in a polled environment.
2. Open the driver using DRV_SST26_Open with the necessary intent.
3. Set an event handler callback using the function DRV_SST26_EventHandlerSet.
4. Request for block operations using the functions, DRV_SST26_Erase, DRV_SST26_Write, DRV_SST26_Read and DRV_SST26_EraseWrite with the appropriate parameters.
5. Wait for event handler callback to occur and check the status of the block operation using the callback function parameter of type DRV_SST26_ EVENT.
6. After performing the required block operations, the client can close the driver using the function , DRV_SST25VF020B_Close .

Example:

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST26_COMMAND_HANDLE commandHandle;

// drvSST26Handle is the handle returned by the DRV_SST26_Open
// function. Client registers an event handler with driver. This is done once.

DRV_SST26_EventHandlerSet(drvSST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_Read(drvSST26Handle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event
    // handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // Operation completed successfully.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Media Interface Functions

Reading the Device Geometry

The application can call the DRV_SST26_GeometryGet function to obtain the geometry of the flash device. The geometry indicates the number of read, write and erase regions, number of blocks per region and the size of each block.

Example:

```
SYS_FS_MEDIA_GEOMETRY * sst26FlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
```

```
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sst26FlashGeometry = DRV_SST26_GeometryGet(sst26OpenHandle1);

readBlockSize  = sst26FlashGeometry->geometryTable->blockSize;
nReadBlocks = sst26FlashGeometry->geometryTable->numBlocks;
nReadRegions = sst26FlashGeometry->numReadRegions;

writeBlockSize  = (sst26FlashGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (sst26FlashGeometry->geometryTable +2)->blockSize;

//The below expression provides the flash memory size.
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Configuring the Library

Macros

	Name	Description
	DRV_SST26_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_SST26_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_SST26_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_SST26_SYS_FS_REGISTER	Register to use with the File system

Description

The SQI Flash Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_SST26_BUFFER_OBJECT_NUMBER Macro

Selects the maximum number of buffer objects

File

drv_sst26_config_template.h

C

```
#define DRV_SST26_BUFFER_OBJECT_NUMBER 5
```

Description

SST26 Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The SST26 Driver can queue up DRV_SST26_BUFFER_OBJECT_NUMBER of read/write/erase requests before return a DRV_SST26_BUFFER_HANDLE_INVALID due to the queue being full. Buffer objects are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SST26_CLIENTS_NUMBER Macro

Selects the maximum number of clients

File

drv_sst26_config_template.h

C

```
#define DRV_SST26_CLIENTS_NUMBER 1
```

Description

SST26 maximum number of clients

This definition selects the maximum number of clients that the SST26 driver can supported at run time. This constant defines the total number of

SST26 driver clients that will be available to all instances of the SST26 driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SST26_INSTANCES_NUMBER Macro

Selects the maximum number of Driver instances that can be supported by the dynamic driver.

File

drv_sst26_config_template.h

C

```
#define DRV_SST26_INSTANCES_NUMBER 1
```

Description

SST26 Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_SST26_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_sst26_config_template.h

C

```
#define DRV_SST26_SYS_FS_REGISTER
```

Description

SST26 Driver Register with File System

Specifying this macro enables the SST26 driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the SST26 driver is to be used with the File System.

Building the Library

This section lists the files that are available in the SQI Flash Driver Library.

Description

This section list the files that are available in the `/src` folder of the SQI Flash Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/sqi_flash`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sst26/drv_sst26.h`	Header file that exports the SST26VF driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
sst26/src/dynamic/drv_sst26.c	Basic SQI Flash Driver SST26VF implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SQI Flash Driver Library depends on the following modules:

- SQI Driver Library
- Ports System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_SST26_Initialize	Initializes the SST26 instance for the specified driver index
	DRV_SST26_Deinitialize	Deinitializes the specified instance of the SST26 driver module
	DRV_SST26_Status	Gets the current status of the SST26 driver module.
	DRV_SST26_Tasks	Maintains the SST26 driver's internal state machine.

b) Core Client Functions

	Name	Description
	DRV_SST26_Open	Opens the specified SST26 driver instance and returns a handle to it
	DRV_SST26_Close	Closes an opened-instance of the SST26 driver

c) Block Operation Functions

	Name	Description
	DRV_SST26_Erase	Erase the specified number of flash blocks from the specified block start address.
	DRV_SST26_EraseWrite	Erase and Write blocks of data starting from a specified block start address.
	DRV_SST26_Read	Reads blocks of data from the specified block start address.
	DRV_SST26_Write	Writes blocks of data starting at the specified block start address.
	DRV_SST26_CommandStatus	Gets the current status of the command.
	DRV_SST26_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

d) Media Interface Functions

	Name	Description
	DRV_SST26_AddressGet	Returns the SST26 media start address
	DRV_SST26_GeometryGet	Returns the geometry of the device.
	DRV_SST26_IsAttached	Returns the physical attach status of the SST26.
	DRV_SST26_IsWriteProtected	Returns the write protect status of the SST26.

e) Data Types and Constants

	Name	Description
	DRV_SST26_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SST26_COMMAND_STATUS	SST26 Driver command Status
	DRV_SST26_EVENT	Identifies the possible events that can result from a request.
	DRV_SST26_EVENT_HANDLER	Pointer to a SST26 Driver Event handler function
	DRV_SST26_INIT	Defines the data required to initialize or reinitialize the SST26 driver
	DRV_SST26_COMMAND_HANDLE_INVALID	This value defines the SST26 Driver's Invalid Command Handle.
	DRV_SST26_INDEX_0	SST26 driver index definitions
	DRV_SST26_INDEX_1	This is macro DRV_SST26_INDEX_1.

Description

This section describes the API functions of the SQI Flash Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_SST26_Initialize Function

Initializes the SST26 instance for the specified driver index

File

drv_sst26.h

C

```
SYS_MODULE_OBJ DRV_SST26_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the SST26 driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other SST26 routine is called.

This routine should only be called once during system initialization unless DRV_SST26_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to initialize, it will be reported by the DRV_SST26_Status operation. The system must use DRV_SST26_Status to find out when the driver is in the ready state.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST26 Driver.

SYS_MODULE_OBJ  objectHandle;

const DRV_SST26_INIT drvSst26InitData0 =
{
    .sqiDevice = 1,
};

//usage of DRV_SST26_INDEX_0 indicates usage of Flash-related APIs
objectHandle = DRV_SST26_Initialize(DRV_SST26_INDEX_0, (SYS_MODULE_INIT*)&drvSst26InitData0);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SST26_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_SST26_Deinitialize Function

Deinitializes the specified instance of the SST26 driver module

File

drv_sst26.h

C

```
void DRV_SST26_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SST26 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_SST26_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_SST26_Initialize routine

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST26_Initialize
SYS_STATUS          status;

DRV_SST26_Deinitialize(object);

status = DRV_SST26_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know when the driver is
    // deinitialized.
}
```

Function

void DRV_SST26_Deinitialize

(

SYS_MODULE_OBJ object

);

DRV_SST26_Status Function

Gets the current status of the SST26 driver module.

File

drv_sst26.h

C

```
SYS_STATUS DRV_SST26_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations.

SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the SST26 driver module.

Remarks

This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_SST26_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST26_Initialize
SYS_STATUS          SST26Status;

SST26Status = DRV_SST26_Status(object);
else if (SYS_STATUS_ERROR >= SST26Status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST26_Initialize routine

Function

SYS_STATUS DRV_SST26_Status

(

SYS_MODULE_OBJ object

);

DRV_SST26_Tasks Function

Maintains the SST26 driver's internal state machine.

File

drv_sst26.h

C

```
void DRV_SST26_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine maintains the driver's internal state machine. Part of the driver initialization is done in this routine. This routine is responsible for processing the read, write, erase or erasewrite requests queued for the SST26 driver.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_SST26_Initialize routine must have been called for the specified SST26 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST26_Initialize

while (true)
{
    DRV_SST26_Tasks (object);
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST26_Initialize)

Function

void DRV_SST26_Tasks
(
SYS_MODULE_OBJ object
);

b) Core Client Functions

DRV_SST26_Open Function

Opens the specified SST26 driver instance and returns a handle to it

File

drv_sst26.h

C

```
DRV_HANDLE DRV_SST26_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST26_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified SST26 driver instance and provides a handle. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_SST26_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has has already been opened, it cannot be opened exclusively.

Preconditions

Function DRV_SST26_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST26_Open(DRV_SST26_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST26_Open

(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

DRV_SST26_Close Function

Closes an opened-instance of the SST26 driver

File

drv_sst26.h

C

```
void DRV_SST26_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the SST26 driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST26_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST26_Initialize routine must have been called for the specified SST26 driver instance.

DRV_SST26_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST26_Open

DRV_SST26_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SST26_Close
(
const DRV_HANDLE handle
);

c) Block Operation Functions

DRV_SST26_Erase Function

Erase the specified number of flash blocks from the specified block start address.

File

drv_sst26.h

C

```
void DRV_SST26_Erase(const DRV_HANDLE handle, DRV_SST26_COMMAND_HANDLE * commandHandle, uint32_t
blockStart, uint32_t nBlock);
```

Returns

The command handle is returned in the commandHandle argument. It Will be DRV_SST26_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation of flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST26_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the client opened the driver for read only
- if the number of blocks to be erased is either zero or more than the number of blocks actually available
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST26_EVENT_COMMAND_COMPLETE event if the erase operation was successful or DRV_SST26_EVENT_COMMAND_ERROR event if the erase operation was not successful.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE to obtain a valid opened device handle.

Example

```
// Use DRV_SST26_GeometryGet () to find the read region geometry.
// Find the erase block start address from where the number of blocks
// should be erased.
uint32_t blockStart = 0;
uint32_t nBlock = 4;
DRV_SST26_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST26Handle is the handle returned by the DRV_SST26_Open function.

// Client registers an event handler with driver
DRV_SST26_EventHandlerSet(mySST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_Erase( mySST26Handle, &commandHandle, blockStart, nBlock );

if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer queue is processed.

void APP_SST26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // Erase operation completled successfully.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;
```

```
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Erase block start address from where the blocks should be erased.
nBlock	Total number of blocks to be erased.

Function

void DRV_SST26_Erase

(

const DRV_HANDLE handle,

DRV_SST26_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,

uint32_t nBlock

);

DRV_SST26_EraseWrite Function

Erase and Write blocks of data starting from a specified block start address.

File

drv_sst26.h

C

```
void DRV_SST26_EraseWrite(const DRV_HANDLE handle, DRV_SST26_COMMAND_HANDLE * commandHandle, void *
sourceBuffer, uint32_t writeBlockStart, uint32_t nWriteBlock);
```

Returns

The command handle is returned in the commandHandle argument. It Will be DRV_SST26_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function combines the step of erasing a sector and then writing the page. The application can use this function if it wants to avoid having to explicitly delete a sector in order to update the pages contained in the sector.

This function schedules a non-blocking operation to erase and write blocks of data into flash memory. The function returns with a valid command handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST26_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST26_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SST26_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE as a parameter to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// Use DRV_SST26_GeometryGet () to find the write region geometry.
// Find the block address to which data is to be written.
uint32_t blockStart = SST26_BLOCK_ADDRESS_TO_WRITE_TO;
uint32_t nBlock = 2;
DRV_SST26_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST26Handle is the handle returned by the DRV_SST26_Open function.
// Client registers an event handler with driver

DRV_SST26_EventHandlerSet(mySST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_EraseWrite(mySST26Handle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // Operation completled successfully.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return command handle. If NULL, then command handle is not returned.
sourceBuffer	The source buffer containing data to be programmed into SST26 Flash
writeBlockStart	Write block start address where the write should begin.
nWriteBlock	Total number of blocks to be written.

Function

void DRV_SST26_EraseWrite

(

const DRV_HANDLE handle,

DRV_SST26_COMMAND_HANDLE * commandHandle,

void * sourceBuffer,

uint32_t writeBlockStart,

uint32_t nWriteBlock

);

DRV_SST26_Read Function

Reads blocks of data from the specified block start address.

File

drv_sst26.h

C

```
void DRV_SST26_Read(const DRV_HANDLE handle, DRV_SST26_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The command handle is returned in the commandHandle argument. It will be DRV_SST26_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from the flash memory. The function returns with a valid command handle in the commandHandle argument if the request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST26_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the number of blocks to be read is either zero or more than the number of blocks actually available
- if the driver handle is invalid

Remarks

None.

Preconditions

The DRV_SST26_Initialize routine must have been called for the specified SST26 driver instance.

DRV_SST26_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// Use DRV_SST26_GeometryGet () to find the read region geometry.
// Find the block address from which to read data.
uint32_t blockStart = SST26_BLOCK_ADDRESS_TO_READ_FROM;
uint32_t nBlock = 2;
DRV_SST26_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST26Handle is the handle returned by the DRV_SST26_Open function.
// Client registers an event handler with driver

DRV_SST26_EventHandlerSet(mySST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_Read(mySST26Handle, &commandHandle, &myBuffer, blockStart, nBlock);
if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read queued successfully.
}

// Event is received when the command request is processed.

void APP_SST26EventHandler
(
```

```
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // This means the data was transferred.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the command handle
targetBuffer	Buffer into which the data read from the SST26 Flash memory will be placed
blockStart	Read block start address from where the data should be read.
nBlock	Total number of blocks to be read.

Function

void DRV_SST26_Read
(
const DRV_HANDLE handle,
DRV_SST26_COMMAND_HANDLE * commandHandle,
void * targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST26_Write Function

Writes blocks of data starting at the specified block start address.

File

drv_sst26.h

C

```
void DRV_SST26_Write(const DRV_HANDLE handle, DRV_SST26_COMMAND_HANDLE * commandHandle, void *
sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The command handle is returned in the commandHandle argument. It will be DRV_SST26_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid command handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST26_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the source buffer pointer is NULL
- if the client opened the driver for read only

- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST26_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SST26_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

DRV_SST26_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

The flash address location which has to be written, must have be erased before using the DRV_SST26_Erase() routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// Use DRV_SST26_GeometryGet () to find the write region geometry.
// Find the block address to which data is to be written.
uint32_t blockStart = SST26_BLOCK_ADDRESS_TO_WRITE_TO;
uint32_t nBlock = 2;
DRV_SST26_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST26Handle is the handle returned by the DRV_SST26_Open function.
// Client registers an event handler with driver

DRV_SST26_EventHandlerSet(mySST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_Write(mySST26Handle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // This means the data was transferred.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SST26 Flash

blockStart	Write block start address from where the data should be written to.
nBlock	Total number of blocks to be written.

Function

void DRV_SST26_Write
(
const DRV_HANDLE handle,
DRV_SST26_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST26_CommandStatus Function

Gets the current status of the command.

File

drv_sst26.h

C

```
DRV_SST26_COMMAND_STATUS DRV_SST26_CommandStatus(const DRV_HANDLE handle, const DRV_SST26_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_SST26_COMMAND_STATUS value describing the current status of the command. Returns DRV_SST26_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to be polled on. The function may return DRV_SST26_COMMAND_COMPLETED in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive the command completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST26_Initialize() routine must have been called.

The DRV_SST26_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                  handle;         // Returned from DRV_SST26_Open
DRV_SST26_COMMAND_HANDLE      commandHandle;
DRV_SST26_COMMAND_STATUS      status;

status = DRV_SST26_CommandStatus(handle, commandHandle);
if(status == DRV_SST26_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_SST26_COMMAND_STATUS DRV_SST26_CommandStatus
(
const DRV_HANDLE handle,

```
    const       DRV_SST26_COMMAND_HANDLE commandHandle
    );
```

DRV_SST26_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_sst26.h

C

```
void DRV_SST26_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls a read, write, erase or a erasewrite function, it is provided with a handle identifying the command that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST26_COMMAND_HANDLE commandHandle;

// drvSST26Handle is the handle returned by the DRV_SST26_Open function.
// Client registers an event handler with driver. This is done once.

DRV_SST26_EventHandlerSet(drvSST26Handle, APP_SST26EventHandler, (uintptr_t)&myAppObj);

DRV_SST26_Read(drvSST26Handle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SST26_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
```

```
            case DRV_SST26_EVENT_COMMAND_COMPLETE:
                // Operation completled successfully.
                break;

            case DRV_SST26_EVENT_COMMAND_ERROR:
                // Error handling here.
                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST26_EventHandlerSet
(
const DRV_HANDLE handle,
const void * eventHandler,
const uintptr_t context
);

d) Media Interface Functions

DRV_SST26_AddressGet Function

Returns the SST26 media start address

File

drv_sst26.h

C

```
uintptr_t DRV_SST26_AddressGet(const DRV_HANDLE handle);
```

Returns

Start address of the SST26 Media if the handle is valid otherwise NULL.

Description

This function returns the SST26 Media start address.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called to obtain a valid opened device handle.

Example

```
uintptr_t startAddress;
startAddress = DRV_SST26_AddressGet(drvSST26Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

uintptr_t DRV_SST26_AddressGet
(
const DRV_HANDLE handle
);

DRV_SST26_GeometryGet Function

Returns the geometry of the device.

File

drv_sst26.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST26_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST26 Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * sst26FlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sst26FlashGeometry = DRV_SST26_GeometryGet(sst26OpenHandle1);

readBlockSize  = sst26FlashGeometry->geometryTable->blockSize;
nReadBlocks = sst26FlashGeometry->geometryTable->numBlocks;
nReadRegions = sst26FlashGeometry->numReadRegions;

writeBlockSize  = (sst26FlashGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (sst26FlashGeometry->geometryTable +2)->blockSize;

totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_SST26_GeometryGet
(
const DRV_HANDLE handle

);

DRV_SST26_IsAttached Function

Returns the physical attach status of the SST26.

File

drv_sst26.h

C

```
bool DRV_SST26_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns true.

Description

This function returns the physical attach status of the SST26.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isSST26Attached;
isSST26Attached = DRV_SST26_isAttached(drvSST26Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST26_IsAttached

(

const DRV_HANDLE handle

);

DRV_SST26_IsWriteProtected Function

Returns the write protect status of the SST26.

File

drv_sst26.h

C

```
bool DRV_SST26_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

True - If the flash is write protected. False - If the flash is not write protected.

Description

This function returns the write protect status of the SST26.

Remarks

None.

Preconditions

The DRV_SST26_Initialize() routine must have been called for the specified SST26 driver instance.

The DRV_SST26_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isWriteProtected;
isWriteProtected = DRV_SST26_IsWriteProtected(drvSST26Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST26_IsWriteProtected

(

const DRV_HANDLE handle

);

e) Data Types and Constants

DRV_SST26_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_sst26.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST26_COMMAND_HANDLE;
```

Description

SST26 Driver command handle.

A command handle is returned by a call to the Read, Write, Erase or EraseWrite functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST26_COMMAND_STATUS Enumeration

SST26 Driver command Status

File

drv_sst26.h

C

```
typedef enum {
  DRV_SST26_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_SST26_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_SST26_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_SST26_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_SST26_COMMAND_STATUS;
```

Members

Members	Description
DRV_SST26_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_SST26_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_SST26_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer

DRV_SST26_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

SST26 Driver Command Status

Specifies the status of the command for the read, write, erase and erasewrite operations.

Remarks

None.

DRV_SST26_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst26.h

C

```
typedef enum {
  DRV_SST26_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST26_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST26_EVENT;
```

Members

Members	Description
DRV_SST26_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_SST26_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

SST26 Driver Events

This enumeration identifies the possible events that can result from a read, write, erase or erasewrite request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST26_EventHandlerSet function when a request is completed.

DRV_SST26_EVENT_HANDLER Type

Pointer to a SST26 Driver Event handler function

File

drv_sst26.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_SST26_EVENT_HANDLER;
```

Returns

None.

Description

SST26 Driver Event Handler Function Pointer

This data type defines the required function signature for the SST26 event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST26_EVENT_COMMAND_COMPLETE, it means that the requested operation was completed successfully.

If the event is DRV_SST26_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_SST26_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations within this function.

Example

```
void APP_MySst26EventHandler
(
    DRV_SST26_EVENT event,
    DRV_SST26_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST26_EVENT_COMMAND_COMPLETE:
            // Handle the completed buffer.
            break;

        case DRV_SST26_EVENT_COMMAND_ERROR:
        default:
            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase/EraseWrite requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST26_INIT Structure

Defines the data required to initialize or reinitialize the SST26 driver

File

drv_sst26.h

C

```
typedef struct {
  uint8_t sqiDevice;
} DRV_SST26_INIT;
```

Members

Members	Description
uint8_t sqiDevice;	SQI Device Index.

Description

SST26 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the SST26 driver.

Remarks

Not all initialization features are available for all devices. Please refer to the specific device data sheet to determine availability.

DRV_SST26_COMMAND_HANDLE_INVALID Macro

This value defines the SST26 Driver's Invalid Command Handle.

File

drv_sst26.h

C

```
#define DRV_SST26_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SST26 Driver Invalid Command Handle.

This value defines the SST26 Driver's Invalid Command Handle. This value is returned by read/write/erase/erasewrite routines when the command request was not accepted.

Remarks

None.

DRV_SST26_INDEX_0 Macro

SST26 driver index definitions

File

drv_sst26.h

C

```
#define DRV_SST26_INDEX_0 0
```

Description

Driver SST26 Module Index reference

These constants provide SST26 driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_SST26_Initialize and DRV_SST26_Open routines to identify the driver instance in use.

DRV_SST26_INDEX_1 Macro

File

drv_sst26.h

C

```
#define DRV_SST26_INDEX_1 1
```

Description

This is macro DRV_SST26_INDEX_1.

Files

Files

Name	Description
drv_sst26.h	SST26 Driver Interface Definition
drv_sst26_config_template.h	SST26 driver configuration definitions.

Description

This section lists the source and header files used by the SQI Flash Driver Library.

drv_sst26.h

SST26 Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST26_COMMAND_STATUS	SST26 Driver command Status
	DRV_SST26_EVENT	Identifies the possible events that can result from a request.

Functions

	Name	Description
	DRV_SST26_AddressGet	Returns the SST26 media start address
	DRV_SST26_Close	Closes an opened-instance of the SST26 driver
	DRV_SST26_CommandStatus	Gets the current status of the command.
	DRV_SST26_Deinitialize	Deinitializes the specified instance of the SST26 driver module
	DRV_SST26_Erase	Erase the specified number of flash blocks from the specified block start address.
	DRV_SST26_EraseWrite	Erase and Write blocks of data starting from a specified block start address.
	DRV_SST26_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SST26_GeometryGet	Returns the geometry of the device.
	DRV_SST26_Initialize	Initializes the SST26 instance for the specified driver index
	DRV_SST26_IsAttached	Returns the physical attach status of the SST26.
	DRV_SST26_IsWriteProtected	Returns the write protect status of the SST26.
	DRV_SST26_Open	Opens the specified SST26 driver instance and returns a handle to it
	DRV_SST26_Read	Reads blocks of data from the specified block start address.
	DRV_SST26_Status	Gets the current status of the SST26 driver module.
	DRV_SST26_Tasks	Maintains the SST26 driver's internal state machine.
	DRV_SST26_Write	Writes blocks of data starting at the specified block start address.

Macros

	Name	Description
	DRV_SST26_COMMAND_HANDLE_INVALID	This value defines the SST26 Driver's Invalid Command Handle.
	DRV_SST26_INDEX_0	SST26 driver index definitions
	DRV_SST26_INDEX_1	This is macro DRV_SST26_INDEX_1.

Structures

	Name	Description
	DRV_SST26_INIT	Defines the data required to initialize or reinitialize the SST26 driver

Types

	Name	Description
	DRV_SST26_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SST26_EVENT_HANDLER	Pointer to a SST26 Driver Event handler function

Description

SST26 Driver Interface Definition

The SST26 driver provides a simple interface to manage the SST26VF series of SQI Flash Memory connected to Microchip microcontrollers. This file defines the interface definition for the SST26 driver.

File Name

drv_sst26.h

Company

Microchip Technology Inc.

drv_sst26_config_template.h

SST26 driver configuration definitions.

Macros

	Name	Description
	DRV_SST26_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_SST26_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_SST26_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_SST26_SYS_FS_REGISTER	Register to use with the File system

Description

SST26 Driver Configuration Template Header file.

This template file describes all the mandatory and optional configuration macros that are needed for building the SST26 driver. Do not include this file in source code.

File Name

drv_sst26_config_template.h

Company

Microchip Technology Inc.

SRAM Driver Library

This section describes the Static Random Access (SRAM) driver library.

Introduction

The SRAM Media Driver library provides a high-level interface to manage the onboard SRAM as a media

Description

The SRAM Media driver features the following:

- Provides application ready routines to perform block operations on the SRAM media
- Supports multi-client operation
- Provides data transfer events
- Supports blocking mode of operation only

Using the Library

This topic describes the basic architecture of the SRAM Media Driver Library and provides information and examples about how to use it.

Description

Interface Header Files: `drv_sram.h`

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Topics

Name	Description
Abstraction Model	This library provides a low-level abstraction of the SRAM media with a convenient C language interface. This topic describes how that abstraction is modeled in software.
Library Overview	This library provides information about how the driver operates in a system. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SRAM Media driver module.
How the Library Works	This section describes how the SRAM Media Driver Library operates

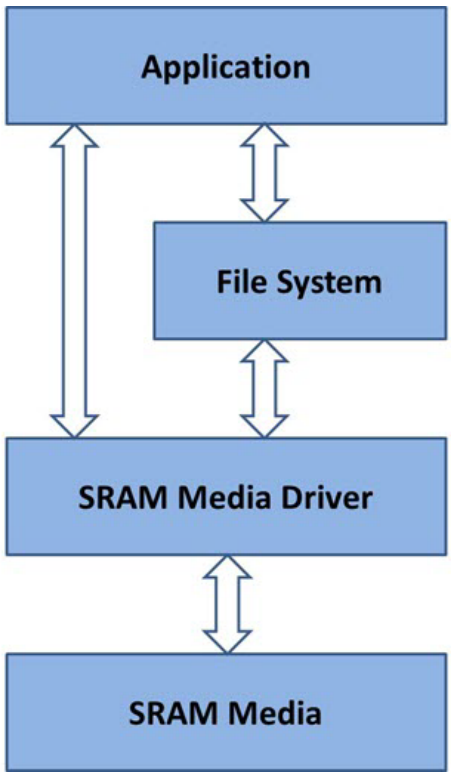
Abstraction Model

This library provides a low-level abstraction of the SRAM media with a convenient C language interface. This section describes how that abstraction is modeled in software.

Description

The SRAM Media driver facilitates the block access to the SRAM media by providing APIs that can be used to perform read/write operations. The end applications can either access the media directly through the driver or through MPLAB Harmony. The following diagram shows the communication between different modules.

SRAM Media Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information about how the SRAM driver operates within a system.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SRAM module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, de-initialization and status functions.
Data Types and Constants	Provides data types and macros.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality
- Media Functionality

System Initialization/Status Functions

The SRAM driver provides the following system functions:

- SRAM driver initialization
- SRAM driver status.

Description

SRAM driver initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization each instance of the SRAM Media driver would be initialized with the following configuration settings passed dynamically at run time using DRV_SRAM_INIT:

- registerWithFs: This controls the registration of the driver with the Harmony File System.
- mediaStartAddress: This indicates the start address of the SRAM media on which driver is to operate.
- sramMediaGeometry: This provides the SRAM media geometry information that the driver will use while performing the read/write operations on the media.

The DRV_SRAM_Initialize function configures and initializes the SRAM Media driver by using the configuration information provided. The function returns an object handle of the type SYS_MODULE_OBJ. This object handle is used by other system interfaces such as DRV_SRAM_Status and DRV_SRAM_Deinitialize.

Example:

```
/*** SRAM Driver Initialization Data ***/
SYS_FS_MEDIA_REGION_GEOMETRY sramMedia0GeometryTable[3] =
{
{
.blockSize = 1,
.numBlocks = (32 * (1024/1)),
},
{
.blockSize = 1,
.numBlocks = (32 * (1024/1)),
},
{
.blockSize = 1,
.numBlocks = (32 * (1024/1)),
}
};

const SYS_FS_MEDIA_GEOMETRY sramMedia0Geometry =
{
.mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING | SYS_FS_MEDIA_READ_IS_BLOCKING,
.numReadRegions = 1,
.numWriteRegions = 1,
.numEraseRegions = 1,
.geometryTable = (SYS_FS_MEDIA_REGION_GEOMETRY *)&sramMedia0GeometryTable
};

extern uint8_t SRAM_MEDIA_0_DATA[];
const DRV_SRAM_INIT drvSram0Init =
{
.registerWithFs = true,
.mediaStartAddress = (uint8_t *)SRAM_MEDIA_0_DATA,
.sramMediaGeometry = (SYS_FS_MEDIA_GEOMETRY
*)&sramMedia0Geometry
};

/* Initialize the SRAM Driver Instance 0 */
sysObj.drvSramObj0 = DRV_SRAM_Initialize(DRV_SRAM_INDEX_0, (SYS_MODULE_INIT *)&drvSram0Init);
```

SRAM driver status

DRV_SRAM_Status() returns the current status of the SRAM Media driver and is called by the Harmony System. The application may not find the need to call this function directly.

Example:

```
SYS_MODULE_OBJ object;
// Returned from DRV_SRAM_Initialize
SYS_STATUS sramStatus;

sramStatus = DRV_SRAM_Status(object);
if (SYS_STATUS_ERROR >= sramStatus)
```

```
{
// Handle error
}
```

Client Core Functions

The SRAM driver provides the following client core functions:

- Opening the driver
- Closing the driver

Description

Opening the driver

For the application to start using an instance of the module, it must call the function DRV_SRAM_Open and obtain a valid driver handle. The application client uses this driver handle to access the driver functions.

For the various options available for I/O INTENT, please refer to Data Types and Constants in the Library Interface section.

Example:

```
handle = DRV_SRAM_Open(DRV_SRAM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
/* Call until the function returns a valid handle. */
}
else
{
/* Do further processing. */
}
```

Closing the driver

This function closes an opened-instance of the SRAM Media driver. This invalidates the driver handle. The application must open the driver again to obtain a valid handle.

Example:

```
DRV_HANDLE handle; // Returned from DRV_SRAM_Open
DRV_SRAM_Close(handle);
```

Client Block Operation Functions

The SRAM driver provides client interfaces to perform operations in terms of blocks.

Description

A block is a unit that represents the minimum amount of data that can be written or read. The block sizes may differ for write and read operations. The DRV_SRAM_GeometryGet function can be used to read out the geometry of the SRAM Media. The geometry indicates the number of read, write, and erase regions; blocks per region; and the size of each block.

The DRV_SRAM_Write and DRV_SRAM_Read functions are used to write and read the data to/from SRAM media. These functions block operations until the requested amount of data is transferred. If an event handler has been registered to receive the driver events, then the event handler will be invoked from within these functions. The driver provides events (DRV_SRAM_EVENT) that indicate the completion of the requests.

The following steps can be performed for a simple block data transfer operation:

1. Make sure the system has completed the necessary initialization of the SRAM driver.
2. Open the driver using DRV_SRAM_Open with the necessary intent.
3. Register an event handler callback by using the function DRV_SRAM_EventHandlerSet.
4. Request for block operation by using the functions DRV_SRAM_Write and DRV_SRAM_Read with the appropriate parameters.
5. Wait for the event handler callback to occur and check the status of the block operation by using the callback function parameter of type DRV_SRAM_ EVENT.
6. After performing the required block operations, the client can close the driver by using the function DRV_SRAM_Close.

Example:

```
uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart = 0;
uint32_t nBlock = 2;
DRV_SRAM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySRAMHandle is the handle returned by the DRV_SRAM_Open
```

```
// function.

DRV_SRAM_EventHandlerSet(
mySRAMHandle,
APP_SRAMEventHandler,
(uintptr_t)&myAppObj);

DRV_SRAM_Read(
mySRAMHandle,
&commandHandle,
&myBuffer,
blockStart,
nBlock);
if(DRV_SRAM_COMMAND_HANDLE_INVALID == commandHandle)
{
// Error handling here
}
else
{
// Read operation completed successfully.
}

// Event is invoked from within the DRV_SRAM_Read function when
// the read operation processing is complete.

void APP_SRAMEventHandler
(
 DRV_SRAM_EVENT event,
 DRV_SRAM_COMMAND_HANDLE commandHandle,
uintptr_t contextHandle
)
{
// contextHandle points to myAppObj.
switch(event)
{
case DRV_SRAM_EVENT_COMMAND_COMPLETE:
// This means the data was transferred.
break;

case DRV_SRAM_EVENT_COMMAND_ERROR:
// Error handling here.
break;

default:
break;
}
}
```

Media Interface Functions

The SRAM driver provides the following media interface function:

- Reading the device geometry

Description

The application can call the DRV_SRAM_GeometryGet function to obtain the geometry of the media device.

Reading the device geometry

The geometry indicates the number of read, write, and erase regions; the number of blocks per region; and the size of each block.

Example:

```
SYS_FS_MEDIA_GEOMETRY * sramGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
```

```
uint32_t nReadBlocks, nReadRegions, totalSize;

sramGeometry = DRV_SRAM_GeometryGet(sramOpenHandle1);

readBlockSize = sramGeometry->geometryTable->blockSize;
nReadBlocks = sramGeometry->geometryTable->numBlocks;
nReadRegions = sramGeometry->numReadRegions;

writeBlockSize = (sramGeometry->geometryTable +1)->blockSize;
eraseBlockSize = (sramGeometry->geometryTable +2)->blockSize;
totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Configuring the Library

This section contains related configuration macros.

Description

The configuration of the SRAM driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SRAM driver. Based on the selections made, the SRAM driver may support the selected features. These configuration settings apply to all instances of the SRAM driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the SRAM library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/sram/`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_sram.h`	This is the SRAM library's interface header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_sram.c`	This file contains the source code for the dynamic implementation of the SRAM driver.
`/config/drv_sram_config_template.h`	This file contains configuration macros for the SRAM driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Library Interface

This section describes the Application Programming Interface (API) functions of the SRAM driver library.

Refer to each section for a detailed description.

a) System Functions

	Name	Description
	DRV_SRAM_AddressGet	Returns the SRAM media start address
	DRV_SRAM_Close	Closes an opened-instance of the SRAM driver
	DRV_SRAM_CommandStatus	Gets the current status of the command.
	DRV_SRAM_Deinitialize	Deinitializes the specified instance of the SRAM driver module
	DRV_SRAM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when an operation has completed.
	DRV_SRAM_GeometryGet	Returns the geometry of the device.
	DRV_SRAM_Initialize	Initializes the SRAM instance for the specified driver index.
	DRV_SRAM_IsAttached	Returns the physical attach status of the SRAM.
	DRV_SRAM_IsWriteProtected	Returns the write protect status of the SRAM.
	DRV_SRAM_Open	Opens the specified SRAM driver instance and returns a handle to it
	DRV_SRAM_Read	Reads blocks of data from the specified block start address.
	DRV_SRAM_Status	Gets the current status of the SRAM driver module.
	DRV_SRAM_Write	Writes blocks of data starting from the specified block start address of the SRAM media.

c) Data Types and Constants

	Name	Description
	DRV_SRAM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SRAM_COMMAND_STATUS	Specifies the status of the command for the read and write operations.
	DRV_SRAM_EVENT	Identifies the possible events that can result from a request.
	DRV_SRAM_EVENT_HANDLER	Pointer to a SRAM Driver Event handler function
	DRV_SRAM_INIT	Defines the data required to initialize the SRAM driver
	_DRV_SRAM_H	This is macro _DRV_SRAM_H.
	DRV_SRAM_COMMAND_HANDLE_INVALID	This value defines the SRAM Driver's Invalid Command Handle.
	DRV_SRAM_INDEX_0	SRAM driver index definitions
	DRV_SRAM_INDEX_1	This is macro DRV_SRAM_INDEX_1.

a) System Functions

DRV_SRAM_AddressGet Function

Returns the SRAM media start address

File

drv_sram.h

C

```
uintptr_t DRV_SRAM_AddressGet(const DRV_HANDLE handle);
```

Returns

Start address of the SRAM Media if the handle is valid otherwise NULL.

Description

This function returns the SRAM Media start address.

Remarks

None.

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

The DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
uintptr_t startAddress;
```

```
startAddress = DRV_SRAM_AddressGet(drvSRAMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open

Function

uintptr_t DRV_SRAM_AddressGet
(
const DRV_HANDLE handle
);

DRV_SRAM_Close Function

Closes an opened-instance of the SRAM driver

File

drv_sram.h

C

```
void DRV_SRAM_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the SRAM driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SRAM_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SRAM_Initialize routine must have been called for the specified SRAM driver instance.

DRV_SRAM_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SRAM_Open

DRV_SRAM_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SRAM_Close
(
const DRV_HANDLE handle
);

DRV_SRAM_CommandStatus Function

Gets the current status of the command.

File

drv_sram.h

C

```
DRV_SRAM_COMMAND_STATUS DRV_SRAM_CommandStatus(const DRV_HANDLE handle, const DRV_SRAM_COMMAND_HANDLE
```

```
commandHandle);
```

Returns

A DRV_SRAM_COMMAND_STATUS value describing the current status of the command. Returns DRV_SRAM_COMMAND_COMPLETED if the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to be polled on. The function may return DRV_SRAM_COMMAND_COMPLETED in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another read or write request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive read or write operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SRAM_Initialize() routine must have been called.

The DRV_SRAM_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                  handle;         // Returned from DRV_SRAM_Open
DRV_SRAM_COMMAND_HANDLE     commandHandle;
DRV_SRAM_COMMAND_STATUS     status;

status = DRV_SRAM_CommandStatus(handle, commandHandle);
if(status == DRV_SRAM_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_SRAM_COMMAND_STATUS DRV_SRAM_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SRAM_COMMAND_HANDLE commandHandle

);

DRV_SRAM_Deinitialize Function

Deinitializes the specified instance of the SRAM driver module

File

drv_sram.h

C

```
void DRV_SRAM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SRAM driver module, disabling its operation. Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_SRAM_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_SRAM_Initialize routine

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SRAM_Initialize
SYS_STATUS          status;

DRV_SRAM_Deinitialize(object);

status = DRV_SRAM_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know when the driver is
    // deinitialized.
}
```

Function

void DRV_SRAM_Deinitialize

(

SYS_MODULE_OBJ object

);

DRV_SRAM_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when an operation has completed.

File

drv_sram.h

C

```
void DRV_SRAM_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when an operation has completed. When a client calls a read or a write function, it is provided with a handle identifying the read/write request. The driver will pass this handle back to the client by calling "eventHandler" function when the operation has completed.

The event handler should be set before the client performs any read or write operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

The DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SRAM_COMMAND_HANDLE commandHandle;

// drvSRAMHandle is the handle returned by the DRV_SRAM_Open function.
```

```
// Client registers an event handler with driver. This is done once.

DRV_SRAM_EventHandlerSet(drvSRAMHandle, APP_SRAMEventHandler, (uintptr_t)&myAppObj);

DRV_SRAM_Read(drvSRAMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SRAM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.
void APP_SRAMEventHandler
(
    DRV_SRAM_EVENT event,
    DRV_SRAM_COMMAND_HANDLE handle,
    uintptr_t context
)
{
    // The context handle was set to an application specific object. It is
    // now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SRAM_EVENT_COMMAND_COMPLETE:
            // This means the data was transferred.
            break;

        case DRV_SRAM_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SRAM_EventHandlerSet
(
const DRV_HANDLE handle,
const void * eventHandler,
const uintptr_t context
);

DRV_SRAM_GeometryGet Function

Returns the geometry of the device.

File

drv_sram.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SRAM_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SRAM memory:

- Media Property
- Number of Read/Write/Erase regions
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

The DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * sramGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalSize;

sramGeometry = DRV_SRAM_GeometryGet(sramOpenHandle1);

readBlockSize  = sramGeometry->geometryTable->blockSize;
nReadBlocks = sramGeometry->geometryTable->numBlocks;
nReadRegions = sramGeometry->numReadRegions;

writeBlockSize  = (sramGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (sramGeometry->geometryTable +2)->blockSize;

totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_SRAM_GeometryGet

(

const DRV_HANDLE handle

);

DRV_SRAM_Initialize Function

Initializes the SRAM instance for the specified driver index.

File

drv_sram.h

C

```
SYS_MODULE_OBJ DRV_SRAM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the SRAM driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other SRAM routine is called.

This routine should only be called once during system initialization unless DRV_SRAM_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. The system must use DRV_SRAM_Status to find out when the driver is in the ready state.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SRAM Driver.

SYS_MODULE_OBJ  objectHandle;

SYS_FS_MEDIA_REGION_GEOMETRY gSramGeometryTable[3] =
{
    {
        // Read Region Geometry
        .blockSize = 512,
        .numBlocks = (DRV_SRAM_MEDIA_SIZE * (1024/512)),
    },
    {
        // Write Region Geometry
        .blockSize = 512,
        .numBlocks = ((DRV_SRAM_MEDIA_SIZE * (1024/512))
    },
    {
        // Erase Region Geometry
        .blockSize = 512,
        .numBlocks = ((DRV_SRAM_MEDIA_SIZE * (1024/512))
    }
};

const SYS_FS_MEDIA_GEOMETRY gSramGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,

    // Number of read, write and erase entries in the table
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = &gSramGeometryTable
};

// SRAM Driver Initialization Data
const DRV_SRAM_INIT drvSramInit =
{
    .mediaStartAddress = DRV_SRAM_MEDIA_START_ADDRESS,
    .sramMediaGeometry = &gSramGeometry
};

objectHandle = DRV_SRAM_Initialize(DRV_SRAM_INDEX_0, (SYS_MODULE_INIT*)&drvSRAMInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized.
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SRAM_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

);

DRV_SRAM_IsAttached Function

Returns the physical attach status of the SRAM.

File

drv_sram.h

C

```
bool DRV_SRAM_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns true.

Description

This function returns the physical attach status of the SRAM.

Remarks

None.

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

The DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The SRAM media is always attached and so the below always returns true.
bool isSRAMAttached;
isSRAMAttached = DRV_SRAM_isAttached(drvSRAMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open

Function

bool DRV_SRAM_IsAttached

(

const DRV_HANDLE handle

);

DRV_SRAM_IsWriteProtected Function

Returns the write protect status of the SRAM.

File

drv_sram.h

C

```
bool DRV_SRAM_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Always returns false.

Description

This function returns the physical attach status of the SRAM. This function always returns false.

Remarks

None.

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

The DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The SRAM media is treated as always writeable.
bool isWriteProtected;
isWriteProtected = DRV_SRAM_IsWriteProtected(drvSRAMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SRAM_IsWriteProtected

(

const DRV_HANDLE handle

);

DRV_SRAM_Open Function

Opens the specified SRAM driver instance and returns a handle to it

File

drv_sram.h

C

```
DRV_HANDLE DRV_SRAM_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SRAM_CLIENTS_NUMBER

is insufficient

- if the client is trying to open the driver but driver has been opened

exclusively by another client

- if the client is trying to open the driver exclusively, but has

already been opened in a non exclusive mode by another client.

- if the driver hardware instance being opened is invalid

Description

This routine opens the specified SRAM driver instance and provides a handle. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_SRAM_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has has already been opened, it cannot be opened exclusively.

Preconditions

DRV_SRAM_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SRAM_Open(DRV_SRAM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened

intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SRAM_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

DRV_SRAM_Read Function

Reads blocks of data from the specified block start address.

File

drv_sram.h

C

```
void DRV_SRAM_Read(const DRV_HANDLE handle, DRV_SRAM_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SRAM_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine reads blocks of data from the specified block start address. This operation is blocking and returns with the required data in the target buffer. If a event handler has been registered to receive the driver events then the event handler will be called from within this function. The function returns DRV_SRAM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the driver state is not ready
- if the target buffer pointer is NULL
- if the number of blocks to be read is zero or more than the actual number

of blocks available

- if the client opened the driver in write only mode

Remarks

None.

Preconditions

The DRV_SRAM_Initialize routine must have been called for the specified SRAM driver instance.

DRV_SRAM_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart = 0;
uint32_t nBlock = 2;
DRV_SRAM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySRAMHandle is the handle returned by the DRV_SRAM_Open function.

DRV_SRAM_EventHandlerSet(mySRAMHandle, APP_SRAMEventHandler, (uintptr_t)&myAppObj);
DRV_SRAM_Read(mySRAMHandle, &commandHandle, &myBuffer, blockStart, nBlock);
if(DRV_SRAM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read operation completed successfully.
```

```
}

// Event is invoked from within the DRV_SRAM_Read function when the read
// operation processing is complete.

void APP_SRAMEventHandler
(
    DRV_SRAM_EVENT event,
    DRV_SRAM_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SRAM_EVENT_COMMAND_COMPLETE:
            // This means the data was transferred.
            break;

        case DRV_SRAM_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
targetBuffer	Buffer into which the data read from the SRAM memory will be placed
blockStart	SRAM media's start block address from where the read should begin.
nBlock	Total number of blocks to be read.

Function

```
void DRV_SRAM_Read
(
const       DRV_HANDLE handle,
    DRV_SRAM_COMMAND_HANDLE * commandHandle,
void * targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);
```

DRV_SRAM_Status Function

Gets the current status of the SRAM driver module.

File

drv_sram.h

C

```
SYS_STATUS DRV_SRAM_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations.
SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the SRAM driver module.

Remarks

None.

Preconditions

Function DRV_SRAM_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SRAM_Initialize
SYS_STATUS          SRAMStatus;

SRAMStatus = DRV_SRAM_Status(object);
if (SRAMStatus == SYS_STATUS_READY)
{
    // Driver is ready to process read/write operations.
}
else
{
    // Driver is not ready.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SRAM_Initialize routine

Function

SYS_STATUS DRV_SRAM_Status

(

SYS_MODULE_OBJ object

);

DRV_SRAM_Write Function

Writes blocks of data starting from the specified block start address of the SRAM media.

File

drv_sram.h

C

```
void DRV_SRAM_Write(const DRV_HANDLE handle, DRV_SRAM_COMMAND_HANDLE * commandHandle, void * sourceBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SRAM_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine writes blocks of data starting at the specified block start address. This operation is blocking and returns after having written the data. If a event handler has been registered to receive the driver events then the event handler will be called from within this function. The function returns DRV_SRAM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the driver state is not ready
- if the source buffer pointer is NULL
- if the number of blocks to be written is zero or more than the actual

number of blocks available

- if the client opened the driver in read only mode

Remarks

None

Preconditions

The DRV_SRAM_Initialize() routine must have been called for the specified SRAM driver instance.

DRV_SRAM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart = 2;
uint32_t nBlock = 2;
DRV_SRAM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySRAMHandle is the handle returned by the DRV_SRAM_Open function.
// Client registers an event handler with driver

DRV_SRAM_EventHandlerSet(mySRAMHandle, APP_SRAMEventHandler, (uintptr_t)&myAppObj);
DRV_SRAM_Write(mySRAMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SRAM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Write completed successfully.
}

// Event is received from within the DRV_SRAM_Write function when the
// buffer is processed.

void APP_SRAMEventHandler
(
    DRV_SRAM_EVENT event,
    DRV_SRAM_COMMAND_HANDLE commandHandle,
    uintptr_t contextHandle
)
{
    // contextHandle points to myAppObj.
    switch(event)
    {
        case DRV_SRAM_EVENT_COMMAND_COMPLETE:
            // This means the data was transferred.
            break;

        case DRV_SRAM_EVENT_COMMAND_ERROR:
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SRAM memory
blockStart	Start block address of SRAM media from where the write should begin.
nBlock	Total number of blocks to be written.

Function

void DRV_SRAM_Write

(

```
const       DRV_HANDLE handle,
    DRV_SRAM_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);
```

c) Data Types and Constants

DRV_SRAM_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_sram.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SRAM_COMMAND_HANDLE;
```

Description

SRAM Driver command handle.

A command handle is returned by a call to the Read or Write functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SRAM_COMMAND_STATUS Enumeration

Specifies the status of the command for the read and write operations.

File

drv_sram.h

C

```
typedef enum {
  DRV_SRAM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_SRAM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_SRAM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_SRAM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_SRAM_COMMAND_STATUS;
```

Members

Members	Description
DRV_SRAM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_SRAM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_SRAM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SRAM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

SRAM Driver Command Status

SRAM Driver command Status

This type specifies the status of the command for the read and write operations.

Remarks

None.

DRV_SRAM_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sram.h

C

```
typedef enum {
  DRV_SRAM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SRAM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_SRAM_EVENT;
```

Members

Members	Description
DRV_SRAM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_SRAM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

SRAM Driver Events

This enumeration identifies the possible events that can result from a read or a write request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SRAM_EventHandlerSet function when a request is completed.

DRV_SRAM_EVENT_HANDLER Type

Pointer to a SRAM Driver Event handler function

File

drv_sram.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_SRAM_EVENT_HANDLER;
```

Returns

None.

Description

SRAM Driver Event Handler Function Pointer

This data type defines the required function signature for the SRAM event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event callbacks from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SRAM_EVENT_COMMAND_COMPLETE, it means that the read or write operation was completed successfully.

If the event is DRV_SRAM_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_SRAM_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write request.

The event handler function executes in the driver's context. It is recommended of the application to not perform process intensive or blocking operations within this function.

Example

```
void APP_MySramEventHandler
(
    DRV_SRAM_EVENT event,
    DRV_SRAM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SRAM_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SRAM_EVENT_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write requests
context	Value identifying the context of the application that registered the event handling function

DRV_SRAM_INIT Structure

Defines the data required to initialize the SRAM driver

File

drv_sram.h

C

```
typedef struct {
  bool registerWithFs;
  uint8_t* mediaStartAddress;
  const SYS_FS_MEDIA_GEOMETRY * sramMediaGeometry;
} DRV_SRAM_INIT;
```

Members

Members	Description
bool registerWithFs;	Flag to indicate if the driver is to be registered with the file system.
uint8_t* mediaStartAddress;	SRAM Media start address. The driver treats this address as block 0 • address for read and write operations.
const SYS_FS_MEDIA_GEOMETRY * sramMediaGeometry;	SRAM Media geometry object.

Description

SRAM Driver Initialization Data

This data type defines the data required to initialize the SRAM driver.

Remarks

None.

_DRV_SRAM_H Macro

File

drv_sram.h

C

```
#define _DRV_SRAM_H
```

Description

This is macro _DRV_SRAM_H.

DRV_SRAM_COMMAND_HANDLE_INVALID Macro

This value defines the SRAM Driver's Invalid Command Handle.

File

drv_sram.h

C

```
#define DRV_SRAM_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SRAM Driver Invalid Command Handle.

This value defines the SRAM Driver Invalid Command Handle. This value is returned by read/write routines when the command request is not accepted.

Remarks

None.

DRV_SRAM_INDEX_0 Macro

SRAM driver index definitions

File

drv_sram.h

C

```
#define DRV_SRAM_INDEX_0 0
```

Description

Driver SRAM Module Index reference

These constants provide SRAM driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_SRAM_Initialize and DRV_SRAM_Open routines to identify the driver instance in use.

DRV_SRAM_INDEX_1 Macro

File

drv_sram.h

C

```
#define DRV_SRAM_INDEX_1 1
```

Description

This is macro DRV_SRAM_INDEX_1.

Files

Files

Name	Description
drv_sram.h	SRAM Driver Interface Definition

Description

This section will list only the library's interface header file(s).

drv_sram.h

SRAM Driver Interface Definition

Enumerations

	Name	Description
	DRV_SRAM_COMMAND_STATUS	Specifies the status of the command for the read and write operations.
	DRV_SRAM_EVENT	Identifies the possible events that can result from a request.

Functions

	Name	Description
	DRV_SRAM_AddressGet	Returns the SRAM media start address
	DRV_SRAM_Close	Closes an opened-instance of the SRAM driver
	DRV_SRAM_CommandStatus	Gets the current status of the command.
	DRV_SRAM_Deinitialize	Deinitializes the specified instance of the SRAM driver module
	DRV_SRAM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when an operation has completed.
	DRV_SRAM_GeometryGet	Returns the geometry of the device.
	DRV_SRAM_Initialize	Initializes the SRAM instance for the specified driver index.
	DRV_SRAM_IsAttached	Returns the physical attach status of the SRAM.
	DRV_SRAM_IsWriteProtected	Returns the write protect status of the SRAM.
	DRV_SRAM_Open	Opens the specified SRAM driver instance and returns a handle to it
	DRV_SRAM_Read	Reads blocks of data from the specified block start address.
	DRV_SRAM_Status	Gets the current status of the SRAM driver module.
	DRV_SRAM_Write	Writes blocks of data starting from the specified block start address of the SRAM media.

Macros

	Name	Description
	_DRV_SRAM_H	This is macro _DRV_SRAM_H.
	DRV_SRAM_COMMAND_HANDLE_INVALID	This value defines the SRAM Driver's Invalid Command Handle.
	DRV_SRAM_INDEX_0	SRAM driver index definitions
	DRV_SRAM_INDEX_1	This is macro DRV_SRAM_INDEX_1.

Structures

	Name	Description
	DRV_SRAM_INIT	Defines the data required to initialize the SRAM driver

Types

	Name	Description
	DRV_SRAM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SRAM_EVENT_HANDLER	Pointer to a SRAM Driver Event handler function

Description

SRAM Driver Interface Definition

The SRAM driver provides a simple interface to manage the SRAM Memory on Microchip microcontrollers. This file defines the interface definition for the SRAM driver.

File Name

drv_sram.h

Company

Microchip Technology Inc.

Timer Driver Library

This section describes the Timer Driver Library.

Introduction

This library provides an interface to manage the Timer module on the Microchip family of microcontrollers during different modes of operation.

Description

Timers are useful for generating accurate time based periodic interrupts for software application or real time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions and accurate hardware delays.

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Using the Library

This topic describes the basic architecture of the Timer Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_tmr.h`

The interface to the Timer Driver Library is defined in the `drv_tmr.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

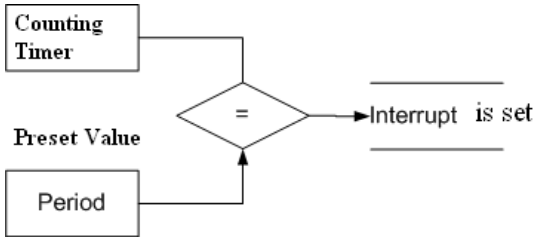
Abstraction Model

The Timer Driver abstracts the hardware by providing the capability to register callback functions to the application.

Description

Abstraction Model

The abstraction model of the Timer Driver is explained in the following diagram:



The core functionality of the Timer allows access to both the counter and the period values.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Timer Driver Library.

Library Interface Section	Description
Configuration	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section. These macros enable different features or modes of the timer peripheral.
System Interaction Functions	Provides interfaces to system layer to initialize, deinitialize and reinitialize the module. This section also describes functions to query the status of the module.
Core Functions	Provides interfaces for core functionality of the driver.
Alarm Functions	Provides interfaces to handle alarm features, if alarm functionality is enabled.
Period Functions	Provides interfaces to control the periodicity of the timers.
Counter Control Functions	Provides interfaces to update the counter values.
Miscellaneous Functions	Provides interfaces to get the version information, timer tick and operating frequencies.

How the Library Works

The library provides interfaces to support:

- System Interaction
- Sync Mode Setup
- Period Modification
- Counter Modification
- Client Core Functionality
- Client Alarm Functionality (optional function, enabled using configuration options)
- Other Optional Functionality (enabled using configuration options)

Note: Any code segment pertaining to the driver interfaces will work for both the static or dynamic configurations. It is not necessary to modify the code to move from one configuration to the other (i.e., from static or dynamic or static-multi).

System Interaction

This section describes Timer initialization and reinitialization.

Description

Initialization and Reinitialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized.

The DRV_TMR_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_TMR_Deinitialize and DRV_TMR_Status, DRV_TMR_Tasks.

Example: Timer Initialization

```
DRV_TMR_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       tmrStatus;

// populate the TMR init configuration structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale         = TMR_PRESCALE_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_2;
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;
init.asyncWriteEnable = false;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);

if (object == SYS_MODULE_OBJ_INVALID)
{
    // Handle error
}
```

Deinitialization

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will never block for hardware Timer access.

Status

Timer status is available to query the module state after initialization and reinitialization.

Tasks Routine

The interface DRV_TMR_Tasks needs to be called by the system task service in a polled environment and in an interrupt-based system.

Example: Polling

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
        return 0;

    while (1)
    {
        DRV_TMR_Tasks (object);
    }
}
```

Example: Interrupt

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
        return 0;

    while (1);
}

/* Sample interrupt routine not specific to any device family */
void ISR T1Interrupt(void)
{
    //Call the Timer Tasks routine
    DRV_TMR_Tasks(object);
}
```

Client Interaction

This section describes general client operation.

Description

General Client Operation

For the application to begin using an instance of the Timer module, it must call the DRV_TMR_Open function. This provides the configuration required to open the Timer instance for operation.

The Timer Driver supports only the 'DRV_IO_INTENT_EXCLUSIVE' IO_INTENT.

Example:

```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}
```

The function DRV_TMR_Close closes an already opened instance of the Timer Driver, invalidating the handle. DRV_TMR_Open must have been

called to obtain a valid opened device handle.

Example:
```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

/*...*/

DRV_TMR_Close( handle );
```

The client has the option to check the status through the interface DRV_TMR_ClientStatus.

Example:
```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

if ( DRV_TMR_CLIENT_STATUS_READY != DRV_TMR_ClientStatus( handle ) )
   return 0;
```

Modification

This section describes Period modification for the different types of Timers (i.e., 16-/32-bit).

Description

These set of functions help modify the Timer periodicity at the client level.

Period Modification

Periodicity of Timer (16/32-bit) can be modified using DRV_TMR_AlarmPeriodSet and the current period can be obtained using DRV_TMR_AlarmPeriodGet.

Example:
```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */

/* Update the new period */
DRV_TMR_AlarmPeriodSet( handle, 0xC350);
```

Counter Modification

This section describes counter modification for the different types of Timers (i.e., 8-/16-/32-bit).

Description

These set of functions help modify the initial value of the Timer counters to help adjust any errors in the periodicity.

Counter Modification

The Timer initial value can be modified using DRV_TMR_CounterValueSet and the current counter value can be obtained using DRV_TMR_CounterValueGet.

Example:
```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */

/* Update the counter value */
/* Following code updates the initial value from 0x0000 to 0x0010
   to cover up any error in the previously set periodicity */

DRV_TMR_CounterValueSet( handle, 0x0010);
```

Core Functionality

This section describes core functionality of the Timer Driver.

Description

Core functionality provides an extremely basic interface for the driver operation.

Applications using the Timer core functionality need to perform the following:

1. The system should have completed the necessary initialization and DRV_TMR_Tasks should be called in a polled/interrupt environment.
2. Open_the driver using DRV_TMR_Open. The Timer Driver only supports exclusive access.
3. The Timer can be updated using DRV_TMR_AlarmPeriodSet. The previously set value can be retrieved using DRV_TMR_AlarmPeriodGet.
4. Start the driver using DRV_TMR_Start.
5. Poll for the elapsed alarm status using DRV_TMR_AlarmHasElapsed.
6. The client will be able to stop the started Timer instance using DRV_TMR_Stop at any time and will be able to close it using DRV_TMR_Close when it is no longer required.

Example:

```
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );
DRV_TMR_Start (handle);
unsigned int alarmCount = 0;
while (1)
{
    if (true == DRV_TMR_AlarmHasElapsed (handle))
    {
        alarmCount++;
        // Do something
    }
}
```

Notes:

1. The user needs to stop the Timer before any updates on the counter or period and restart it later.
2. The Timer alarm count gets reset after any call to DRV_TMR_AlarmHasElapsed.
3. The Timer alarm status remains unchanged if the user stops the timer and restarts later.

Alarm Functionality

This section describes the Timer Driver alarm functionality.

Description

The Timer Driver provides alarm functionality.

Applications using the Timer alarm functionality, need to perform the following:

1. The system should have completed the necessary initialization and DRV_TMR_Tasks should be running in either a polled environment or in an interrupt environment.
2. Open the driver using DRV_TMR_Open. The Timer Driver supports exclusive access only.
3. Configure the alarm using DRV_TMR_AlarmRegister.
4. Start the driver using DRV_TMR_Start.
5. If a callback is supplied, the Timer Driver will call the callback function when the alarm expires.
6. The client will be able to stop the started Timer module instance using DRV_TMR_Stop at any time and will be able to close it using DRV_TMR_Close when it is no longer required.
7. The client can deregister the callback by using DRV_TMR_AlarmDeregister.

Example:

```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
/* Configure the timer alarm feature */
uint32_t myFreq = 1000; // 1KHz
uint32_t clkFreq = DRV_TMR_CounterFrequencyGet(tmrHandle);   // timer running frequency

// calculate the divider needed
uint32_t divider = clkFreq / myFreq;

// Start the alarm
if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, CallBackFreq ))
```

```
{
    // divider value could not be obtain;
    // handle the error
    //
    return;
}

DRV_TMR_Start (handle);

// The driver tasks function calls the client registered callback after the alarm expires.
void CallBackFreq (uintptr_t context, uint32_t alarmCount)
{
    // Do something specific on an alarm event trigger
}
```

Optional Interfaces

This section describes additional/optional client interfaces.

Description

Additional/Optional client interfaces include the following:

Get Operating Frequency

The function DRV_TMR_CounterFrequencyGet provides the client with the information on the Timer operating frequency.

Example:

```
DRV_HANDLE handle;
uint32_t freq;

/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

freq = DRV_TMR_OperatingFrequencyGet (handle);
```

Example Usage of the Timer Driver

This section describes typical usage of the Timer Driver for various Timer modules in polling/interrupt advanced/core modes.

Description

The user can pass NULL to the driver initialize interface. However, the respective configuration parameters need to be configured in the correct manner.

Example:

```
//Polled mode under 32-bit count mode for a PIC32 device using the alarm feature
SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLKSOURCE_INTERNAL;
init.prescale         = TMR_PRESCALE_TX_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_3;
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;init.asyncWriteEnable = false;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_CLIENT_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, AlarmCallback ))
{
    // divider value could not be obtain;
```

```
    // handle the error
}

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_CLIENT_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void AlarmCallback (uintptr_t context, uint32_t alarmCount)
{
  // Do something specific on an alarm event trigger
}
```

Configuring the Library

Macros

	Name	Description
	DRV_TMR_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_TMR_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_TMR_CLOCK_PRESCALER	Sets the default timer driver clock prescaler.
	DRV_TMR_MODE	Sets the default timer driver clock operating mode.
	DRV_TMR_MODULE_ID	Sets the default timer module ID to be used by the timer driver.
	DRV_TMR_MODULE_INIT	Sets the default module init value for the timer driver.
	DRV_TMR_INTERRUPT_SOURCE	Sets the default timer driver clock interrupt source
	DRV_TMR_ASYNC_WRITE_ENABLE	Controls Asynchronous Write mode of the Timer.
	DRV_TMR_CLOCK_SOURCE	Sets the default timer driver clock source.
	DRV_TMR_CLIENTS_NUMBER	Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

Description

The configuration of the Timer Driver Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Timer Driver Library build. Based on the selections made here and the system setup, the Timer Driver may support the selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere in the application-specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

DRV_TMR_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INSTANCES_NUMBER 5
```

Description

Hardware instances support

This definition sets up the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None

DRV_TMR_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INTERRUPT_MODE true
```

Description

TMR Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_TMR_CLOCK_PRESCALER Macro

Sets the default timer driver clock prescaler.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLOCK_PRESCALER (TMR_PRESCALE_VALUE_256)
```

Description

Default timer driver clock prescaler

This macro sets the default timer driver clock prescaler.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODE Macro

Sets the default timer driver clock operating mode.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODE (DRV_TMR_OPERATION_MODE_16_BIT)
```

Description

Default timer driver clock operating mode

This macro sets the default timer driver clock operating mode.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODULE_ID Macro

Sets the default timer module ID to be used by the timer driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODULE_ID (TMR_ID_2)
```

Description

Default timer driver index

This macro sets the default timer module ID to be used by the timer driver.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODULE_INIT Macro

Sets the default module init value for the timer driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODULE_INIT (SYS_MODULE_POWER_RUN_FULL)
```

Description

Default module init object configuration

This macro sets the default module init value for the timer driver.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_INTERRUPT_SOURCE Macro

Sets the default timer driver clock interrupt source

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INTERRUPT_SOURCE (INT_SOURCE_TIMER_2)
```

Description

Default timer driver clock interrupt source

This macro sets the default timer driver clock interrupt source

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_ASYNC_WRITE_ENABLE Macro

Controls Asynchronous Write mode of the Timer.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_ASYNC_WRITE_ENABLE false
```

Description

TMR Asynchronous write mode configuration

This macro controls the Asynchronous Write mode of the Timer. This macro accepts the following values:

- true - Configures the Timer to enable asynchronous write control
- false - Configures the Timer to disable asynchronous write control
- DRV_CONFIG_NOT_SUPPORTED - When the feature is not supported on the instance.

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

DRV_TMR_CLOCK_SOURCE Macro

Sets the default timer driver clock source.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLOCK_SOURCE (DRV_TMR_CLKSOURCE_INTERNAL)
```

Description

Default timer driver clock source

This macro sets the default timer driver clock source.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLIENTS_NUMBER 1
```

Description

Client instances support

This definition sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

Remarks

Currently each client is required to get exclusive access to the timer module. Therfore the DRV_TMR_CLIENTS_NUMBER should always be set to 1.

Building the Library

This section lists the files that are available in the Timer Driver Library.

Description

This section list the files that are available in the `\src` folder of the Timer Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/tmr`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_tmr.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_tmr_dynamic.c`	Basic Timer driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Timer Driver Library depends on the following modules:

- Clock System Service Library
- Interrupt System Service Library
- Interrupt Peripheral Library
- Device Control System Service Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Initialize	Initializes the Timer driver. **Implementation:** Static/Dynamic
	DRV_TMR_Status	Provides the current status of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic
	DRV_TMR_ClockSet	Sets the timers clock by selecting the source and prescaler. **Implementation:** Dynamic
	DRV_TMR_GateModeSet	Enables the Gate mode. **Implementation:** Dynamic

b) Core Functions

	Name	Description
	DRV_TMR_ClientStatus	Gets the status of the client operation. **Implementation:** Dynamic
	DRV_TMR_Close	Closes an opened instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TMR_Start	Starts the Timer counting. **Implementation:** Static/Dynamic
	DRV_TMR_Stop	Stops the Timer from counting. **Implementation:** Static/Dynamic

c) Alarm Functions

	Name	Description
	DRV_TMR_AlarmHasElapsed	Provides the status of Timer's period elapse. **Implementation:** Dynamic
	DRV_TMR_AlarmDisable	Disables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmEnable	Re-enables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodGet	Provides the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodSet	Updates the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmRegister	Sets up an alarm. **Implementation:** Dynamic

d) Counter Control Functions

	Name	Description
	DRV_TMR_CounterFrequencyGet	Provides the Timer input frequency. **Implementation:** Dynamic
	DRV_TMR_CounterClear	Clears the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueGet	Reads the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueSet	Updates the Timer's counter register. **Implementation:** Static/Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_TMR_GateModeClear	Enables the Gate mode. **Implementation:** Dynamic
	DRV_TMR_PrescalerGet	This function gets the currently selected prescaler. **Implementation:** Dynamic
	DRV_TMR_OperationModeGet	This function gets the currently selected operation mode. **Implementation:** Dynamic
	DRV_TMR_DividerRangeGet	Returns the Timer divider values. **Implementation:** Dynamic

f) Data Types and Constants

Name	Description
DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.
DRV_TMR_INIT	Defines the Timer driver initialization data.
DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
DRV_TMR_DIVIDER_RANGE	This data structure specifies the divider values that can be obtained by the timer module.
DRV_TMR_OPERATION_MODE	Lists the operation modes available for timer driver.
DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.
DRV_TMR_INDEX_0	Timer driver index definitions
DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
DRV_TMR_INDEX_6	This is macro DRV_TMR_INDEX_6.
DRV_TMR_INDEX_7	This is macro DRV_TMR_INDEX_7.
DRV_TMR_INDEX_8	This is macro DRV_TMR_INDEX_8.

	DRV_TMR_INDEX_9	This is macro DRV_TMR_INDEX_9.
	DRV_TMR_INDEX_10	This is macro DRV_TMR_INDEX_10.
	DRV_TMR_INDEX_11	This is macro DRV_TMR_INDEX_11.

Description

This section describes the functions of the Timer Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_TMR_Deinitialize Function

Deinitializes the specified instance of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the Timer driver, disabling its operation (and any hardware). All internal data is invalidated.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TMR_Status operation. The system has to use DRV_TMR_Status to find out when the module is in the ready state.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      tmrObject;     //  Returned from DRV_TMR_Initialize
SYS_STATUS          tmrStatus;

DRV_TMR_Deinitialize ( tmrObject );

tmrStatus = DRV_TMR_Status ( tmrObject );

if ( SYS_MODULE_UNINITIALIZED == tmrStatus )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Function

void DRV_TMR_Deinitialize (SYS_MODULE_OBJ object)

DRV_TMR_Initialize Function

Initializes the Timer driver.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
SYS_MODULE_OBJ DRV_TMR_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to DRV_TMR_Deinitialize, DRV_TMR_Tasks and DRV_TMR_Status functions.

Description

This function initializes the Timer driver, making it ready for clients to open and use it.

Remarks

This function must be called before any other Timer driver function is called.

This function should only be called once during system initialization unless DRV_TMR_Deinitialize is called to deinitialize the driver instance.

This function will NEVER block for hardware access. The system must use DRV_TMR_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
DRV_TMR_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the timer initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = DRV_TMR_CLKSOURCE_INTERNAL;
init.prescale         = TMR_PRESCALE_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_2;
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;
init.asyncWriteEnable = false;

// Do something

objectHandle = DRV_TMR_Initialize ( DRV_TMR_INDEX_0, (SYS_MODULE_INIT*)&init );

if ( SYS_MODULE_OBJ_INVALID == objectHandle )
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_TMR_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT * const init

)

DRV_TMR_Status Function

Provides the current status of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
SYS_STATUS DRV_TMR_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is initialized and ready for operation

Description

This function provides the current status of the Timer driver.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

The this operation can be used to determine when any of the driver's module level operations has completed.

Once the status operation returns SYS_STATUS_READY, the driver is ready for operation.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TMR_Initialize
SYS_STATUS          tmrStatus;

tmrStatus = DRV_TMR_Status ( object );

else if ( SYS_STATUS_ERROR >= tmrStatus )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Function

SYS_STATUS DRV_TMR_Status (SYS_MODULE_OBJ object)

DRV_TMR_Tasks Function

Maintains the driver's state machine.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine and processes the timer events..

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TMR_Initialize

while (true)
{
    DRV_TMR_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TMR_Initialize)

Function

void DRV_TMR_Tasks (SYS_MODULE_OBJ object)

DRV_TMR_ClockSet Function

Sets the timers clock by selecting the source and prescaler.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_ClockSet(DRV_HANDLE handle, DRV_TMR_CLK_SOURCES clockSource, TMR_PRESCALE preScale);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the clockSource and/or prescaler are not supported

Description

This function sets the timer clock by selecting the source and prescaler. The clock sources are device specific, refer device datasheet for supported clock sources. If unsupported clock source is passed then the behaviour of this function is unpredictable.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_ClockSet ( tmrHandle, DRV_TMR_CLKSOURCE_INTERNAL, TMR_PRESCALE_VALUE_256 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
clockSource	Clock source of the timer
preScale	Timer's Prescaler divisor

Function

bool DRV_TMR_ClockSet
(
DRV_HANDLE handle,
DRV_TMR_CLK_SOURCES clockSource,
TMR_PRESCALE preScale
)

DRV_TMR_GateModeSet Function

Enables the Gate mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_GateModeSet(DRV_HANDLE handle);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the gate mode is not supported

Description

This function enables the Gated mode of Timer. User can measure the duration of an external signal in this mode. Once the Gate mode is enabled, Timer will start on the raising edge of the external signal. It will keep counting until the next falling edge.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_GateModeSet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_GateModeSet (DRV_HANDLE handle)

b) Core Functions

DRV_TMR_ClientStatus Function

Gets the status of the client operation.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_CLIENT_STATUS DRV_TMR_ClientStatus(DRV_HANDLE handle);
```

Returns

None

Description

This function gets the status of the recently completed client level operation.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_CLIENT_STATUS tmrDrvStatus;

tmrDrvStatus = DRV_TMR_ClientStatus ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_TMR_CLIENT_STATUS DRV_TMR_ClientStatus (DRV_HANDLE handle)

DRV_TMR_Close Function

Closes an opened instance of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the Timer driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling DRV_TMR_Open before the caller may use the driver again.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Close (DRV_HANDLE handle)

DRV_TMR_Open Function

Opens the specified Timer driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_HANDLE DRV_TMR_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This function opens the specified Timer driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. Timer driver does not support multiple clients. If two tasks want to use the timer, one should wait until the other one gets closed.

Remarks

The handle returned is valid until the DRV_TMR_Close function is called.

This function will NEVER block waiting for hardware.

If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID.

The Timer driver does not support DRV_IO_INTENT_SHARED. Only exclusive access is supported for now.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TMR_Open ( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if ( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_TMR_Open

(

const SYS_MODULE_INDEX index,

const DRV_IO_INTENT intent

)

DRV_TMR_Start Function

Starts the Timer counting.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_Start(DRV_HANDLE handle);
```

Returns

- true - if the operation succeeded
- false - the supplied handle is invalid or the client doesn't have the needed parameters to run (alarm callback and period)

Description

This function starts the Timer counting.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Timer parameters must have been set by a call to DRV_TMR_AlarmRegister.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmRegister(tmrHandle, 0x100, true, 0, myTmrCallback);
DRV_TMR_Start ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_Start (DRV_HANDLE handle)

DRV_TMR_Stop Function

Stops the Timer from counting.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Stop(DRV_HANDLE handle);
```

Returns

None.

Description

This function stops the running Timer from counting.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open
```

```
DRV_TMR_Stop ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Stop (DRV_HANDLE handle)

c) Alarm Functions

DRV_TMR_AlarmHasElapsed Function

Provides the status of Timer's period elapse.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_AlarmHasElapsed(DRV_HANDLE handle);
```

Returns

Number of times timer has elapsed since the last call.

Description

This function returns the number of times Timer's period has elapsed since last call to this API has made. On calling this API, the internally maintained counter will be cleared and count will be started again from next elapse.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      tmrHandle;  // Returned from DRV_TMR_Open
bool            elapseStatus;
SYS_MODULE_OBJ  tmrObject // Returned by DRV_TMR_Initialize
unsigned int    appInternalTime = 0;

Sys_Tasks()
{
    //Timer task will be called from ISR

    APP_TimeUpdate_Task();

    //Other Tasks
}

void APP_TimeUpdate_Task ( void )
{
    //We will not miss a count even though we are late
    appInternalTime += DRV_TMR_AlarmHasElapsed ( tmrHandle );
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from the DRV_TMR_Open

Function

unsigned int DRV_TMR_AlarmHasElapsed (DRV_HANDLE handle)

DRV_TMR_AlarmDisable Function

Disables an alarm signal.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_AlarmDisable(DRV_HANDLE handle);
```

Returns

The current status of the alarm:

- true if the alarm was currently enabled
- false if the alarm was currently disabled

Description

This function allows the client to disable an alarm generation. Use DRV_TMR_AlarmEnable to re-enable.

Remarks

When the driver operates in interrupts this call resolves to a device interrupt disable.

Do NOT disable the timer except for very short periods of time. If the time that the interrupt is disabled is longer than a wrap around period and the interrupt is missed, the hardware has no means of recovering and the resulting timing will be inaccurate.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

A client alarm must be active.

Example

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open

Function

bool DRV_TMR_AlarmDisable (DRV_HANDLE handle);

DRV_TMR_AlarmEnable Function

Re-enables an alarm signal.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmEnable(DRV_HANDLE handle, bool enable);
```

Returns

None

Description

This function allows the client to re-enable an alarm after it has been disabled by a DRV_TMR_AlarmDisable call.

Remarks

When the driver operates in interrupts this call resolves to a device interrupt re-enable.

Preconditions

The DRV_TMR_Initialize function must have been called. DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
enable	boolean to enable the current callback

Function

void DRV_TMR_AlarmEnable (DRV_HANDLE handle, bool enable);

DRV_TMR_AlarmDeregister Function

Removes a previously set alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmDeregister(DRV_HANDLE handle);
```

Returns

None.

Description

This function removes a previously set alarm.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

DRV_TMR_AlarmRegister function must have been called before.

Example

```
// Example of a key debounce check

static unsigned int lastReadKey, readKey, keyCount, globalKeyState;
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

void keyPressDetect ()
{
    // Calculate the count to be passed on from the clock input
    DRV_TMR_AlarmRegister ( tmrHandle, 0xFF00, true, DebounceCheck );
}

void DebounceCheck ( uintptr_t context )
{
    readKey = AppReadKey();

    if ( readKey != lastReadKey )
    {
        lastReadKey = readKey;
        keyCount = 0;
    }
    else
    {
        if ( keyCount > 20 )
        {
```

```
            globalKeyState = readKey;
            DRV_TMR_AlarmDeregister ( tmrHandle );
        }
        keyCount++;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_AlarmDeregister (DRV_HANDLE handle)

DRV_TMR_AlarmPeriodGet Function

Provides the Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_AlarmPeriodGet(DRV_HANDLE handle);
```

Returns

Timer period value:

- a 16 bit value if the timer is configured in 16 bit mode
- a 32 bit value if the timer is configured in 32 bit mode

Description

This function gets the Timer's period.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
uint32_t period;

period = DRV_TMR_AlarmPeriodGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_AlarmPeriodGet (DRV_HANDLE handle)

DRV_TMR_AlarmPeriodSet Function

Updates the Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmPeriodSet(DRV_HANDLE handle, uint32_t value);
```

Returns

None.

Description

This function updates the Timer's period.

Remarks

- The period value will be truncated to a 16 bit value if the timer is

configured in 16 bit mode.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmPeriodSet ( handle, 0x1000 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	Period value • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode

Function

void DRV_TMR_AlarmPeriodSet (DRV_HANDLE handle, uint32_t value)

DRV_TMR_AlarmRegister Function

Sets up an alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_AlarmRegister(DRV_HANDLE handle, uint32_t divider, bool isPeriodic, uintptr_t context,
DRV_TMR_CALLBACK callBack);
```

Returns

- true - if the call succeeded
- false - the obtained divider could not be obtained or the passed handle was invalid

Description

This function sets up an alarm, allowing the client to receive a callback from the driver when the timer counter reaches zero. Alarms can be one-shot or periodic. A periodic alarm will reload the timer and generate alarm until stopped. The alarm frequency is: DRV_TMR_CounterFrequencyGet() / divider;

Remarks

The divider value will be truncated to a 16 bit value if the timer is configured in 16 bit mode. The timer should be started using DRV_TMR_Start API to get a callback.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

divider value has to be within the timer divider range (see DRV_TMR_DividerSpecGet).

Example

```
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_16_BIT

void setupTask ()
{
    DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

    uint32_t myFreq = 1000; // 1KHz
    uint32_t clkFreq = DRV_TMR_CounterFrequencyGet(tmrHandle);   // timer running frequency

    // calculate the divider needed
    uint32_t divider = clkFreq / myFreq;

    // Start the alarm
    if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, CallBackFreq ))
    {
        // divider value could not be obtain;
        // handle the error
        //
    }
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
divider	The value to divide the timer clock source to obtain the required alarm frequency. • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode
isPeriodic	Flag indicating whether the alarm should be one-shot or periodic.
context	A reference, call back function will be called with the same reference.
callBack	A call back function which will be called on time out.

Function

bool DRV_TMR_AlarmRegister
(
    DRV_HANDLE handle,
uint32_t divider,
bool isPeriodic,
uintptr_t context,
    DRV_TMR_CALLBACK callBack
)

d) Counter Control Functions

DRV_TMR_CounterFrequencyGet Function

Provides the Timer input frequency.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_CounterFrequencyGet(DRV_HANDLE handle);
```

Returns

32-bit value corresponding to the running frequency. If Timer clock source is external, then this function returns 0.

Description

This function provides the Timer input frequency. Input frequency is the clock to the Timer register and it is considering the prescaler divisor.

Remarks

On most processors, the Timer's base frequency is the same as the peripheral bus clock.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
uint32_t   clkFreqHz;

clkFreqHz = DRV_TMR_CounterFrequencyGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_CounterFrequencyGet (DRV_HANDLE handle)

DRV_TMR_CounterClear Function

Clears the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterClear(DRV_HANDLE handle);
```

Returns

None.

Description

This function clears the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_CounterClear (DRV_HANDLE handle)

DRV_TMR_CounterValueGet Function

Reads the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_CounterValueGet(DRV_HANDLE handle);
```

Returns

Timer current period:

- a 16 bit value if the timer is configured in 16 bit mode
- a 32 bit value if the timer is configured in 32 bit mode

Description

This function returns the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValueSet ( tmrHandle, ( 0xFFFF - 0x1000 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValueGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_CounterValueGet (DRV_HANDLE handle)

DRV_TMR_CounterValueSet Function

Updates the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterValueSet(DRV_HANDLE handle, uint32_t counterPeriod);
```

Returns

None.

Description

This function updates the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
counterPeriod	counter period value • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode

Function

void DRV_TMR_CounterValueSet (DRV_HANDLE handle, uint32_t counterPeriod)

e) Miscellaneous Functions

DRV_TMR_GateModeClear Function

Enables the Gate mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_GateModeClear(DRV_HANDLE handle);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the gate mode is not supported

Description

This function enables the Gated mode of Timer. User can measure the duration of an external signal in this mode. Once the Gate mode is enabled, Timer will start on the raising edge of the external signal. It will keep counting until the next falling edge.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_GateModeClear ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_GateModeClear (DRV_HANDLE handle)

DRV_TMR_PrescalerGet Function

This function gets the currently selected prescaler.

Implementation: Dynamic

File

drv_tmr.h

C

```
TMR_PRESCALE DRV_TMR_PrescalerGet(DRV_HANDLE handle);
```

Returns

Timer prescaler.

Description

This function gets the currently selected prescaler.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
TMR_PRESCALE preScale;

preScale = DRV_TMR_PrescalerGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

TMR_PRESCALE DRV_TMR_PrescalerGet (DRV_HANDLE handle)

DRV_TMR_OperationModeGet Function

This function gets the currently selected operation mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_OPERATION_MODE DRV_TMR_OperationModeGet(DRV_HANDLE handle);
```

Returns

A DRV_TMR_OPERATION_MODE value showing how the timer is currently configured. DRV_TMR_OPERATION_MODE_NONE is returned for an invalid client handle.

Description

This function gets the currently selected 16/32 bit operation mode.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_OPERATION_MODE operMode;

operMode = DRV_TMR_OperationModeGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_TMR_OPERATION_MODE DRV_TMR_OperationModeGet(DRV_HANDLE handle)

DRV_TMR_DividerRangeGet Function

Returns the Timer divider values.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_OPERATION_MODE DRV_TMR_DividerRangeGet(DRV_HANDLE handle, DRV_TMR_DIVIDER_RANGE* pDivRange);
```

Returns

- A DRV_TMR_OPERATION_MODE value showing how the timer is currently configured. The pDivRange is updated with the supported range values.
- DRV_TMR_OPERATION_MODE_NONE for invalid client handle

Description

This function provides the Timer operating mode and divider range.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_OPERATION_MODE  timerMode;
DRV_TMR_DIVIDER_RANGE   timerRange;

DRV_TMR_DividerRangeGet(handle, &timerRange);
```

```
uint32_t   clkFreqHz = DRV_TMR_CounterFrequencyGet ( tmrHandle );

uint32_t    maxFreqHz = clkFreqHz / timerRange.dividerMin;
uint32_t    minFreqHz = clkFreqHz / timerRange.dividerMax;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
pDivRange	Address to store the timer divider range.

Function

DRV_TMR_OPERATION_MODE DRV_TMR_DividerRangeGet
(
DRV_HANDLE handle,
DRV_TMR_DIVIDER_RANGE* pDivRange
)

f) Data Types and Constants

DRV_TMR_CALLBACK Type

Pointer to a Timer driver callback function data type.

File

drv_tmr.h

C

```
typedef void (* DRV_TMR_CALLBACK)(uintptr_t context, uint32_t alarmCount);
```

Description

Timer Driver Callback Function Pointer

This data type defines a pointer to a Timer driver callback function.

Remarks

Useful only when timer alarm callback support is enabled by defining the DRV_TMR_ALARM_ENABLE configuration option.

DRV_TMR_INIT Structure

Defines the Timer driver initialization data.

File

drv_tmr.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  TMR_MODULE_ID tmrId;
  DRV_TMR_CLK_SOURCES clockSource;
  TMR_PRESCALE prescale;
  INT_SOURCE interruptSource;
  DRV_TMR_OPERATION_MODE mode;
  bool asyncWriteEnable;
} DRV_TMR_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization.
TMR_MODULE_ID tmrId;	Identifies timer hardware module (PLIB-level) ID
DRV_TMR_CLK_SOURCES clockSource;	Clock Source select.
TMR_PRESCALE prescale;	Prescaler Selection from the processor enumeration

INT_SOURCE interruptSource;	Interrupt Source for TMR module. If 'DRV_TMR_OPERATION_MODE_32_BIT' flag is selected the interrupt will be generated by the 2nd timer of the pair, the odd numbered one.
DRV_TMR_OPERATION_MODE mode;	Select 16/32 bit operation mode. 32 bit mode will combine two 16 bit timer modules to form a 32 bit one. This is usually only necessary for very long delays.
bool asyncWriteEnable;	Asynchronous write enable configuration. If true the asynchronous write is enabled. For timers that do not support this feature the value is ignored

Description

Timer Driver Initialize Data

This data type defines data required to initialize the Timer driver.

Remarks

Not all initialization features are available on all devices.

DRV_TMR_CLIENT_STATUS Enumeration

Identifies the client-specific status of the Timer driver

File

drv_tmr.h

C

```
typedef enum {
  DRV_TMR_CLIENT_STATUS_INVALID,
  DRV_TMR_CLIENT_STATUS_BUSY,
  DRV_TMR_CLIENT_STATUS_READY,
  DRV_TMR_CLIENT_STATUS_RUNNING
} DRV_TMR_CLIENT_STATUS;
```

Members

Members	Description
DRV_TMR_CLIENT_STATUS_INVALID	Driver is invalid (or unopened) state
DRV_TMR_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_TMR_CLIENT_STATUS_READY	Ready, no operations running
DRV_TMR_CLIENT_STATUS_RUNNING	Timer started and running, processing transactions

Description

Timer Driver Client Status

This enumeration identifies the client-specific status of the Timer driver.

Remarks

None.

DRV_TMR_DIVIDER_RANGE Structure

This data structure specifies the divider values that can be obtained by the timer module.

File

drv_tmr.h

C

```
typedef struct {
  uint32_t dividerMin;
  uint32_t dividerMax;
  uint32_t dividerStep;
} DRV_TMR_DIVIDER_RANGE;
```

Members

Members	Description
uint32_t dividerMin;	The minimum divider value that the timer module can obtain
uint32_t dividerMax;	The maximum divider value that the timer module can obtain
uint32_t dividerStep;	The divider step value, between 2 divider values Should be 1 for most timers

Description

Timer Driver divider operating specification

This data structure specifies the divider values that can be obtained by the timer hardware.

Remarks

None.

DRV_TMR_OPERATION_MODE Enumeration

Lists the operation modes available for timer driver.

File

drv_tmr.h

C

```
typedef enum {
  DRV_TMR_OPERATION_MODE_NONE,
  DRV_TMR_OPERATION_MODE_16_BIT,
  DRV_TMR_OPERATION_MODE_32_BIT
} DRV_TMR_OPERATION_MODE;
```

Members

Members	Description
DRV_TMR_OPERATION_MODE_NONE	The timer module operating mode none/invalid
DRV_TMR_OPERATION_MODE_16_BIT	The timer module operates in 16 bit mode
DRV_TMR_OPERATION_MODE_32_BIT	The timer module operates in 32 bit mode This will combine two 16 bit timer modules

Description

Timer Driver Operation mode

This enumeration lists all the available operation modes that are valid for the timer hardware.

Remarks

Not all modes are available on all devices.

DRV_TMR_INDEX_COUNT Macro

Number of valid Timer driver indices.

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_COUNT TMR_NUMBER_OF_MODULES
```

Description

Timer Driver Module Index Count

This constant identifies Timer driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

DRV_TMR_INDEX_0 Macro

Timer driver index definitions

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_0 0
```

Description

Timer Driver Module Index Numbers

These constants provide Timer driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_TMR_Initialize and DRV_TMR_Open functions to identify the driver instance in use.

DRV_TMR_INDEX_1 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_1 1
```

Description

This is macro DRV_TMR_INDEX_1.

DRV_TMR_INDEX_2 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_2 2
```

Description

This is macro DRV_TMR_INDEX_2.

DRV_TMR_INDEX_3 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_3 3
```

Description

This is macro DRV_TMR_INDEX_3.

DRV_TMR_INDEX_4 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_4 4
```

Description

This is macro DRV_TMR_INDEX_4.

DRV_TMR_INDEX_5 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_5 5
```

Description

This is macro DRV_TMR_INDEX_5.

DRV_TMR_INDEX_6 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_6 6
```

Description

This is macro DRV_TMR_INDEX_6.

DRV_TMR_INDEX_7 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_7 7
```

Description

This is macro DRV_TMR_INDEX_7.

DRV_TMR_INDEX_8 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_8 8
```

Description

This is macro DRV_TMR_INDEX_8.

DRV_TMR_INDEX_9 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_9 9
```

Description

This is macro DRV_TMR_INDEX_9.

DRV_TMR_INDEX_10 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_10 10
```

Description

This is macro DRV_TMR_INDEX_10.

DRV_TMR_INDEX_11 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_11 11
```

Description

This is macro DRV_TMR_INDEX_11.

Files

Files

Name	Description
drv_tmr.h	Timer device driver interface header file.
drv_tmr_config_template.h	Timer driver configuration definitions for the template version.

Description

This section lists the source and header files used by the Timer Driver Library.

drv_tmr.h

Timer device driver interface header file.

Enumerations

	Name	Description
	DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
	DRV_TMR_OPERATION_MODE	Lists the operation modes available for timer driver.

Functions

	Name	Description
	DRV_TMR_AlarmDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmDisable	Disables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmEnable	Re-enables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmHasElapsed	Provides the status of Timer's period elapse. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodGet	Provides the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodSet	Updates the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmRegister	Sets up an alarm. **Implementation:** Dynamic

	DRV_TMR_ClientStatus	Gets the status of the client operation. **Implementation:** Dynamic
	DRV_TMR_ClockSet	Sets the timers clock by selecting the source and prescaler. **Implementation:** Dynamic
	DRV_TMR_Close	Closes an opened instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_CounterClear	Clears the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterFrequencyGet	Provides the Timer input frequency. **Implementation:** Dynamic
	DRV_TMR_CounterValueGet	Reads the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueSet	Updates the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_DividerRangeGet	Returns the Timer divider values. **Implementation:** Dynamic
	DRV_TMR_GateModeClear	Enables the Gate mode. **Implementation:** Dynamic
	DRV_TMR_GateModeSet	Enables the Gate mode. **Implementation:** Dynamic
	DRV_TMR_Initialize	Initializes the Timer driver. **Implementation:** Static/Dynamic
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TMR_OperationModeGet	This function gets the currently selected operation mode. **Implementation:** Dynamic
	DRV_TMR_PrescalerGet	This function gets the currently selected prescaler. **Implementation:** Dynamic
	DRV_TMR_Start	Starts the Timer counting. **Implementation:** Static/Dynamic
	DRV_TMR_Status	Provides the current status of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Stop	Stops the Timer from counting. **Implementation:** Static/Dynamic
	DRV_TMR_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_TMR_INDEX_0	Timer driver index definitions
	DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
	DRV_TMR_INDEX_10	This is macro DRV_TMR_INDEX_10.
	DRV_TMR_INDEX_11	This is macro DRV_TMR_INDEX_11.
	DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
	DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
	DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
	DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
	DRV_TMR_INDEX_6	This is macro DRV_TMR_INDEX_6.
	DRV_TMR_INDEX_7	This is macro DRV_TMR_INDEX_7.
	DRV_TMR_INDEX_8	This is macro DRV_TMR_INDEX_8.
	DRV_TMR_INDEX_9	This is macro DRV_TMR_INDEX_9.
	DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.

Structures

	Name	Description
	DRV_TMR_DIVIDER_RANGE	This data structure specifies the divider values that can be obtained by the timer module.
	DRV_TMR_INIT	Defines the Timer driver initialization data.

Types

	Name	Description
	DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.

Description

Timer Device Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer device driver.

File Name

drv_tmr.h

Company

Microchip Technology Inc.

drv_tmr_config_template.h

Timer driver configuration definitions for the template version.

Macros

	Name	Description
	DRV_TMR_ASYNC_WRITE_ENABLE	Controls Asynchronous Write mode of the Timer.
	DRV_TMR_CLIENTS_NUMBER	Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.
	DRV_TMR_CLOCK_PRESCALER	Sets the default timer driver clock prescaler.
	DRV_TMR_CLOCK_SOURCE	Sets the default timer driver clock source.
	DRV_TMR_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_TMR_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_TMR_INTERRUPT_SOURCE	Sets the default timer driver clock interrupt source
	DRV_TMR_MODE	Sets the default timer driver clock operating mode.
	DRV_TMR_MODULE_ID	Sets the default timer module ID to be used by the timer driver.
	DRV_TMR_MODULE_INIT	Sets the default module init value for the timer driver.

Description

Timer Driver Configuration Definitions for the Template Version

These definitions set up the driver for the default mode of operation of the driver.

File Name

drv_tmr_config_template.h

Company

Microchip Technology Inc.

Touch Driver Libraries Help

This section describes the Touch Driver Libraries.

Generic Touch Driver API

This library help section outlines the generic Touch Driver API to be followed by anyone who wants to use a custom created touch driver to go with the MPLAB Harmony framework for their applications.

Description

This generic driver would still be used with the Touch System Service as described by the API. It provides the data structures and functions required for the touch driver to interface with the graphics library as well as the Touch System Services.

The APIs provide routines to read the touch input data from the touch screen. The driver is based on the device notifying the availability of touch input data through external interrupt.

Currently, the API and the system services only supports non-gestural single-fingered touch input.

Library Interface

Functions

	Name	Description
	DRV_TOUCH_Close	Closes an opened instance of an TOUCH module driver.
	DRV_TOUCH_Deinitialize	Deinitializes the index instance of the TOUCH module.
	DRV_TOUCH_Initialize	Initializes hardware and data for the index instance of the TOUCH module.
	DRV_TOUCH_Open	Opens the specified instance of the Touch driver for use and provides an "open-instance" handle.
	DRV_TOUCH_Read	Notifies the driver that there is current touch data to read
	DRV_TOUCH_Reinitialize	
	DRV_TOUCH_Status	Provides the current status of the index instance of the TOUCH module.
	DRV_TOUCH_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic

Data Types and Constants

	Name	Description
	DRV_TOUCH_INIT	Defines the data required to initialize or reinitialize the TOUCH driver
	DRV_TOUCH_PEN_STATE	Identifies the current state of the pen.
	DRV_TOUCH_POSITION_STATUS	Identifies the current status of the current touch point.
	DRV_TOUCH_SAMPLE_POINTS	This is type DRV_TOUCH_SAMPLE_POINTS.
	DRV_TOUCH_INDEX_0	Touch driver index definitions.
	DRV_TOUCH_INDEX_1	This is macro DRV_TOUCH_INDEX_1.
	DRV_TOUCH_INDEX_COUNT	Number of valid TOUCH driver indices.

Description

Functions

DRV_TOUCH_Close Function

Closes an opened instance of an TOUCH module driver.

File

drv_touch.h

C

```
void DRV_TOUCH_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of an TOUCH module driver, making the specified handle invalid.

Preconditions

The DRV_TOUCH_Initialize routine must have been called for the specified TOUCH device instance and the DRV_TOUCH_Status must have returned SYS_STATUS_READY.

DRV_TOUCH_Open must have been called to obtain a valid opened device handle.

Example

```
myTouchHandle = DRV_TOUCH_Open(DRV_TOUCH_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

DRV_TOUCH_Close(myTouchHandle);
```

Parameters

Parameters	Description
drvHandle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_Close (const DRV_HANDLE drvHandle)

DRV_TOUCH_Deinitialize Function

Deinitializes the index instance of the TOUCH module.

File

drv_touch.h

C

```
void DRV_TOUCH_Deinitialize(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

This function deinitializes the index instance of the TOUCH module, disabling its operation (and any hardware for driver modules). It deinitializes only the specified module instance. It also resets all the internal data structures and fields for the specified instance to the default settings.

Preconditions

The DRV_TOUCH_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS   touchstatus;

DRV_TOUCH_Deinitialize(DRV_TOUCH_ID_1);

touchstatus = DRV_TOUCH_Status(DRV_TOUCH_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the TOUCH module to be deinitialized

Function

void DRV_TOUCH_Deinitialize (const SYS_MODULE_ID index)

DRV_TOUCH_Initialize Function

Initializes hardware and data for the index instance of the TOUCH module.

File

drv_touch.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This function initializes hardware for the index instance of the TOUCH module, using the hardware initialization given data. It also initializes any internal driver data structures making the driver ready to be opened.

Preconditions

None.

Example

```
DRV_TOUCH_INIT_DATA         touchInitData;
SYS_STATUS                  touchStatus;

// Populate the touchInitData structure
touchInitData.moduleInit.powerState = SYS_MODULE_POWER_RUN_FULL;
touchInitData.moduleInit.moduleCode = (DRV_TOUCH_INIT_DATA_MASTER | DRV_TOUCH_INIT_DATA_SLAVE);

DRV_TOUCH_Initialize(DRV_TOUCH_ID_1, (SYS_MODULE_INIT*)&touchInitData);
touchStatus = DRV_TOUCH_Status(DRV_TOUCH_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the TOUCH module to be initialized
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and the default initialization is to be used.

Function

void DRV_TOUCH_Initialize (const TOUCH_MODULE_ID index,

const SYS_MODULE_INIT *const data)

DRV_TOUCH_Open Function

Opens the specified instance of the Touch driver for use and provides an "open-instance" handle.

File

drv_touch.h

C

```
DRV_HANDLE DRV_TOUCH_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a value identifying both the caller and the module instance). If an error occurs, the returned value is DRV_HANDLE_INVALID.

Description

This function opens the specified instance of the Touch module for use and provides a handle that is required to use the remaining driver routines.

This function opens a specified instance of the Touch module driver for use by any client module and provides an "open-instance" handle that must be provided to any of the other Touch driver operations to identify the caller and the instance of the Touch driver/hardware module.

Preconditions

The DRV_TOUCH_Initialize routine must have been called for the specified TOUCH device instance and the DRV_TOUCH_Status must have returned SYS_STATUS_READY.

Example

```
DRV_HANDLE                 touchHandle;
DRV_TOUCH_CLIENT_STATUS    touchClientStatus;

touchHandle = DRV_TOUCH_Open(DRV_TOUCH_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == touchHandle)
{
    // Handle open error
}
```

```
touchClientStatus = DRV_TOUCH_ClientStatus(touchHandle);

// Close the device when it is no longer needed.
DRV_TOUCH_Close(touchHandle);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the TOUCH module to be opened.
intent	Flags parameter identifying the intended usage and behavior of the driver. Multiple flags may be ORed together to specify the intended usage of the device. See the DRV_IO_INTENT definition.

Function

DRV_HANDLE DRV_TOUCH_Open (const SYS_MODULE_INDEX index,
const DRV_IO_INTENT intent)

DRV_TOUCH_Read Function

Notifies the driver that there is current touch data to read

File

drv_touch.h

C

```
size_t DRV_TOUCH_Read(DRV_HANDLE drvHandle, void * buffer, size_t size);
```

Description

Notifies the driver that there is current touch data to read

Example

Function

size_t DRV_TOUCH_Read (DRV_HANDLE drvHandle, void *buffer, size_t size)

DRV_TOUCH_Reinitialize Function

File

drv_touch.h

C

```
void DRV_TOUCH_Reinitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT *const data);
```

Returns

None.

Preconditions

The DRV_TOUCH_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_INIT touchInit;
SYS_STATUS      touchStatus;

DRV_TOUCH_Reinitialize(DRV_TOUCH_ID_1, &touchStatus);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the TOUCH module to be reinitialized
data	Pointer to the data structure containing any data necessary to reinitialize the hardware. This pointer may be null if no data is required and default configuration is to be used.

Function

void DRV_TOUCH_Reinitialize(const SYS_MODULE_ID index,
const SYS_MODULE_INIT *const data)

DRV_TOUCH_Status Function

Provides the current status of the index instance of the TOUCH module.

File

drv_touch.h

C

```
SYS_STATUS DRV_TOUCH_Status(const SYS_MODULE_INDEX index);
```

Description

This function provides the current status of the index instance of the TOUCH module.

Preconditions

The DRV_TOUCH_Initialize function should have been called before calling this function.

Function

SYS_STATUS DRV_TOUCH_Status (const TOUCH_MODULE_ID index)

DRV_TOUCH_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_touch.h

C

```
void DRV_TOUCH_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function. This routine decodes the touch input data available.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TOUCH_Initialize routine must have been called for the specified MTCH6301 driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_Initialize)

Function

void DRV_TOUCH_Tasks (SYS_MODULE_OBJ object);

Data Types and Constants

DRV_TOUCH_INIT Structure

Defines the data required to initialize or reinitialize the TOUCH driver

File

drv_touch.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  int touchId;
  SYS_MODULE_OBJ (* drvInitialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
  DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  void (* drvCalibrationSet)(DRV_TOUCH_SAMPLE_POINTS * samplePoints);
  short (* drvTouchGetX)(uint8_t touchNumber);
  short (* drvTouchGetY)(uint8_t touchNumber);
  DRV_TOUCH_POSITION_STATUS (* drvTouchStatus)(const SYS_MODULE_INDEX index);
  void (* drvTouchDataRead)(const SYS_MODULE_INDEX index);
  DRV_TOUCH_PEN_STATE (* drvTouchPenGet)(uint8_t touchNumber);
  INT_SOURCE interruptSource;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
  uint16_t (* pReadFunc)(uint32_t);
  void (* pWriteFunc)(uint16_t, uint32_t);
  void (* pSectorErase)(uint32_t);
  int32_t minTouchDetectDelta;
} DRV_TOUCH_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
int touchId;	ID
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels
uint16_t (* pReadFunc)(uint32_t);	typedef for read function pointer
void (* pWriteFunc)(uint16_t, uint32_t);	typedef for write function pointer

Description

TOUCH Driver Initialization Data

This data type defines the data required to initialize or reinitialize the TOUCH driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_TOUCH_PEN_STATE Type

Identifies the current state of the pen.

File

drv_touch.h

C

```
typedef enum DRV_TOUCH_PEN_STATE@2 DRV_TOUCH_PEN_STATE;
```

Description

TOUCH Controller Driver Pen State

Identifies the current state of the pen reported from a touch event.

Remarks

This enumeration is the return type for the TouchGetPen routine.

DRV_TOUCH_POSITION_STATUS Type

Identifies the current status of the current touch point.

File

drv_touch.h

C

```
typedef enum DRV_TOUCH_POSITION_STATUS@2 DRV_TOUCH_POSITION_STATUS;
```

Description

TOUCH Controller Driver Touch status

Identifies the current status of the current touch point.

Remarks

This enumeration is the return type for the status routine for the current touch point

DRV_TOUCH_SAMPLE_POINTS Type

File

drv_touch.h

C

```
typedef struct DRV_TOUCH_SAMPLE_POINTS@2 DRV_TOUCH_SAMPLE_POINTS;
```

Description

This is type DRV_TOUCH_SAMPLE_POINTS.

DRV_TOUCH_INDEX_0 Macro

Touch driver index definitions.

File

drv_touch.h

C

```
#define DRV_TOUCH_INDEX_0 0
```

Description

Touch Driver Module Index Numbers

These constants provide the Touch driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_TOUCH_Initialize and DRV_TOUCH_Open functions to identify the driver instance in use.

DRV_TOUCH_INDEX_1 Macro

File

drv_touch.h

C

```
#define DRV_TOUCH_INDEX_1 1
```

Description

This is macro DRV_TOUCH_INDEX_1.

DRV_TOUCH_INDEX_COUNT Macro

Number of valid TOUCH driver indices.

File

drv_touch.h

C

```
#define DRV_TOUCH_INDEX_COUNT 1
```

Description

TOUCH Driver Module Index Count

This constant identifies the number of valid TOUCH driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_touch.h	Touch device driver interface file.

Description

drv_touch.h

Touch device driver interface file.

Functions

	Name	Description
	DRV_TOUCH_Close	Closes an opened instance of an TOUCH module driver.
	DRV_TOUCH_Deinitialize	Deinitializes the index instance of the TOUCH module.
	DRV_TOUCH_Initialize	Initializes hardware and data for the index instance of the TOUCH module.
	DRV_TOUCH_Open	Opens the specified instance of the Touch driver for use and provides an "open-instance" handle.
	DRV_TOUCH_Read	Notifies the driver that there is current touch data to read
	DRV_TOUCH_Reinitialize	
	DRV_TOUCH_Status	Provides the current status of the index instance of the TOUCH module.
	DRV_TOUCH_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_TOUCH_INDEX_0	Touch driver index definitions.
	DRV_TOUCH_INDEX_1	This is macro DRV_TOUCH_INDEX_1.
	DRV_TOUCH_INDEX_COUNT	Number of valid TOUCH driver indices.

Structures

	Name	Description
	DRV_TOUCH_INIT	Defines the data required to initialize or reinitialize the TOUCH driver

Types

	Name	Description
	DRV_TOUCH_PEN_STATE	Identifies the current state of the pen.
	DRV_TOUCH_POSITION_STATUS	Identifies the current status of the current touch point.
	DRV_TOUCH_SAMPLE_POINTS	This is type DRV_TOUCH_SAMPLE_POINTS.

Description

Touch Driver Interface

The Touch driver provides a abstraction to all touch drivers.

File Name

drv_touch.h

Company

Microchip Technology Inc.

10-bit ADC Touch Driver Library

This topic describes the 10-bit ADC Touch Driver Library.

Introduction

This library provides an interface to manage the 10-bit ADC Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony 10-bit ADC Touch Driver provides a high-level interface to the 10-bit ADC touch device. This driver provides application routines to read non-gestural single-point touch input data from the touch screen. The 10-bit ADC touch device can notify the availability of touch input data through external interrupt. The 10-bit ADC Touch Driver allows the application to map a controller pin as an external interrupt pin.

Using the Library

This topic describes the basic architecture of the 10-bit ADC Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_adc10bit.h`

The interface to the 10-bit ADC Touch Driver library is defined in the `drv_adc10bit.h` header file. Any C language source (`.c`) file that uses the ADC 10-bit Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the 10-bit ADC Touch Driver module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

Configuring the Library

Macros

	Name	Description
	DRV_ADC10BIT_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_ADC10BIT_CALIBRATION_INSET	Defines the calibration inset.
	DRV_ADC10BIT_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ADC10BIT_INDEX	ADC10BIT static index selection.
	DRV_ADC10BIT_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_ADC10BIT_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ADC10BIT_SAMPLE_POINTS	Defines the sample points.
	DRV_ADC10BIT_TOUCH_DIAMETER	Defines the touch diameter.

Description

The configuration of the 10-bit ADC Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the ADC 10-bit Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the 10-bit ADC Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ADC10BIT_CALIBRATION_DELAY Macro

Defines the calibration delay.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CALIBRATION_DELAY 300
```

Description

ADC10BIT Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_ADC10BIT_CALIBRATION_INSET Macro

Defines the calibration inset.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CALIBRATION_INSET 25
```

Description

ADC10BIT Calibration Inset

This macro defines the calibration inset.

Remarks

None.

DRV_ADC10BIT_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CLIENTS_NUMBER 1
```

Description

ADC10BIT client number

This macro selects the maximum number of clients.

This definition selected the maximum number of clients that the ADC10BIT driver can support at run-time.

Remarks

None.

DRV_ADC10BIT_INDEX Macro

ADC10BIT static index selection.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INDEX DRV_ADC10BIT_INDEX_0
```

Description

ADC10BIT Static Index Selection

This macro specifies the static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_ADC10BIT_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INSTANCES_NUMBER 1
```

Description

ADC10BIT hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_ADC10BIT_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INTERRUPT_MODE false
```

Description

ADC10BIT Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of ADC10BIT operation is desired
- false - Select if polling mode of ADC10BIT operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_ADC10BIT_SAMPLE_POINTS Macro

Defines the sample points.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_SAMPLE_POINTS 4
```

Description

ADC10BIT Sample Points

This macro defines the sample points.

Remarks

None.

DRV_ADC10BIT_TOUCH_DIAMETER Macro

Defines the touch diameter.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_TOUCH_DIAMETER 10
```

Description

ADC10BIT Touch Diameter

This macro defines the touch diameter.

Remarks

None.

Building the Library

This section lists the files that are available in the 10-bit ADC Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the 10-bit ADC Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/adc10bit`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_adc10bit.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/drv_adc10bit.c	Basic 10-bit ADC Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The 10-bit ADC Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_ADC10BIT_CalibrationSet	Loads calibration parameters from Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_Close	Closes an opened instance of the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_Deinitialize	Deinitializes the specified instance of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Initialize	Initializes the 10-bit ADC Driver instance for the specified driver index
	DRV_TOUCH_ADC10BIT_Open	Opens the specified ADC10BIT driver instance and returns a handle to it.
	DRV_TOUCH_ADC10BIT_Status	Provides the current status of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_TOUCH_ADC10BIT_TouchGetRawX	Returns raw x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetRawY	Returns raw y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetX	Returns x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchStoreCalibration	Stores calibration parameters into Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_PositionDetect	None.
	DRV_TOUCH_ADC10BIT_TouchGetY	Returns y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_ADC10BIT_TouchStatus	Returns the status of the current touch input.

b) Data Types and Constants

	Name	Description
	_DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.
	DRV_ADC10BIT_MODULE_ID	This is type DRV_ADC10BIT_MODULE_ID.
	DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	DRV_TOUCH_ADC10BIT_HANDLE	Driver handle.
	DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_ADC10BIT_INDEX_0	ADC10BIT driver index definitions.
	DRV_TOUCH_ADC10BIT_INDEX_1	This is macro DRV_TOUCH_ADC10BIT_INDEX_1.
	DRV_TOUCH_ADC10BIT_INDEX_COUNT	Number of valid ADC10BIT driver indices.

Description

This section describes the API functions of the 10-bit ADC Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_TOUCH_ADC10BIT_CalibrationSet Function

Loads calibration parameters from Non-volatile Memory.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_CalibrationSet(DRV_TOUCH_SAMPLE_POINTS * samplePoints);
```

Returns

None.

Description

This function loads calibration parameters from Non-volatile Memory.

Preconditions

The NVM initialization function must be called before calling this function.

Function

void DRV_TOUCH_ADC10BIT_TouchLoadCalibration(void)

DRV_TOUCH_ADC10BIT_Close Function

Closes an opened instance of the 10-bit ADC Driver.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the 10-bit ADC Driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_ADC10BIT_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called for the specified ADC10BIT driver instance.

DRV_TOUCH_ADC10BIT_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_ADC10BIT_Open

DRV_TOUCH_ADC10BIT_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_ADC10BIT_Close (DRV_HANDLE handle)

DRV_TOUCH_ADC10BIT_Deinitialize Function

Deinitializes the specified instance of the ADC10BIT driver module.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the 10-bit ADC Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_ADC10BIT_Status operation. The system has to use DRV_TOUCH_ADC10BIT_Status to determine when the module is in the ready state.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_TOUCH_ADC10BIT_Initialize
SYS_STATUS          status;

DRV_TOUCH_ADC10BIT_Deinitialize ( object );

status = DRV_TOUCH_ADC10BIT_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_ADC10BIT_Initialize

Function

void DRV_TOUCH_ADC10BIT_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_ADC10BIT_Initialize Function

Initializes the 10-bit ADC Driver instance for the specified driver index

File

drv_adc10bit.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_ADC10BIT_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the 10-bit ADC Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization

data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the 10-bit ADC Driver module ID. For example, driver instance 0 can be assigned to ADC10BIT2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_ADC10BIT_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other ADC10BIT routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_ADC10BIT_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_ADC10BIT_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the ADC10BIT initialization structure
init.spiId              = ADC10BIT_ID_1;

objectHandle = DRV_TOUCH_ADC10BIT_Initialize(DRV_TOUCH_ADC10BIT_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the 10-bit ADC Driver ID. The hardware 10-bit ADC Driver ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_ADC10BIT_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_TOUCH_ADC10BIT_Open Function

Opens the specified ADC10BIT driver instance and returns a handle to it.

File

drv_adc10bit.h

C

```
DRV_HANDLE DRV_TOUCH_ADC10BIT_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_ADC10BIT_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This function opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_USART_Read() and DRV_USART_Write() functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_TOUCH_ADC10BIT_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_ADC10BIT_Open( DRV_TOUCH_ADC10BIT_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_ADC10BIT_Initialize. Please note this is not the SPI id.
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_TOUCH_ADC10BIT_Open (const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT intent)

DRV_TOUCH_ADC10BIT_Status Function

Provides the current status of the ADC10BIT driver module.

File

drv_adc10bit.h

C

```
SYS_STATUS DRV_TOUCH_ADC10BIT_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the ADC10BIT driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_ADC10BIT_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_ADC10BIT_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_ADC10BIT_Initialize

Function

SYS_STATUS DRV_TOUCH_ADC10BIT_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_ADC10BIT_Tasks Function

Maintains the driver's state machine and implements its ISR.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In Interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called for the specified 10-bit ADC Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_ADC10BIT_Initialize

while( true )
{
    DRV_TOUCH_ADC10BIT_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_ADC10BIT_Initialize)

Function

void DRV_TOUCH_ADC10BIT_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_ADC10BIT_TouchGetRawX Function

Returns raw x coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetRawX();
```

Returns

- raw x coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the raw x coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Function

short DRV_TOUCH_ADC10BIT_TouchGetRawX()

DRV_TOUCH_ADC10BIT_TouchGetRawY Function

Returns raw y coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetRawY();
```

Returns

- raw y coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the raw y coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Function

short DRV_TOUCH_ADC10BIT_TouchGetRawY()

DRV_TOUCH_ADC10BIT_TouchGetX Function

Returns x coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetX(uint8_t touchNumber);
```

Returns

- x coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the x coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
touchNumber	touch input index.

Function

short DRV_TOUCH_ADC10BIT_TouchGetX(uint8_t touchNumber)

DRV_TOUCH_ADC10BIT_TouchStoreCalibration Function

Stores calibration parameters into Non-volatile Memory.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_TouchStoreCalibration();
```

Returns

None.

Description

This function stores calibration parameters into Non-volatile Memory.

Remarks

This API is deprecated and its funcationality is handled via SYSTEM_INITIALIZATION

Preconditions

The NVM initialization function must be called before calling this function.

Function

void DRV_TOUCH_ADC10BIT_TouchStoreCalibration(void)

DRV_TOUCH_ADC10BIT_PositionDetect Function

None.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_PositionDetect();
```

Returns

None.

Description

None.

Preconditions

None.

Function

void DRV_TOUCH_ADC10BIT_TouchLoadCalibration(void)

DRV_TOUCH_ADC10BIT_TouchGetY Function

Returns y coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetY(uint8_t touchNumber);
```

Returns

- y coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the y coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
handle	driver client handle.
touchNumber	touch input index.

Function

short DRV_TOUCH_ADC10BIT_TouchGetY(DRV_HANDLE handle, uint8_t touchNumber)

DRV_TOUCH_ADC10BIT_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_ADC10BIT_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_ADC10BIT_TouchStatus Function

Returns the status of the current touch input.

File

drv_adc10bit.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_ADC10BIT_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_ADC10BIT_TouchStatus(const SYS_MODULE_INDEX index)

b) Data Types and Constants

DRV_ADC10BIT_MODULE_ID Enumeration

File

drv_adc10bit.h

C

```
typedef enum {
  ADC10BIT_ID_1 = 0,
  ADC10BIT_NUMBER_OF_MODULES
} DRV_ADC10BIT_MODULE_ID;
```

Description

This is type DRV_ADC10BIT_MODULE_ID.

DRV_TOUCH_ADC10BIT_CLIENT_DATA Structure

Defines the data that can be changed per client.

File

drv_adc10bit.h

C

```
typedef struct _DRV_TOUCH_ADC10BIT_CLIENT_DATA {
} DRV_TOUCH_ADC10BIT_CLIENT_DATA;
```

Description

Macro: ADC10BIT Driver Client Specific Configuration

This data type defines the data can be configured per client. This data can be per client, and overrides the configuration data contained inside of DRV_TOUCH_ADC10BIT_INIT.

Remarks

None.

DRV_TOUCH_ADC10BIT_HANDLE Type

Driver handle.

File

drv_adc10bit.h

C

```
typedef uintptr_t DRV_TOUCH_ADC10BIT_HANDLE;
```

Description

Macro: ADC10BIT Driver Handle

Touch screen controller interfacing with the 10-bit Analog-to-Digital (ADC) converter device.

Remarks

None

DRV_TOUCH_ADC10BIT_INIT Structure

Defines the data required to initialize or reinitialize the 10-bit ADC Driver.

File

drv_adc10bit.h

C

```
typedef struct _DRV_TOUCH_ADC10BIT_INIT {
  SYS_MODULE_INIT moduleInit;
  DRV_ADC10BIT_MODULE_ID adc10bitId;
} DRV_TOUCH_ADC10BIT_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
DRV_ADC10BIT_MODULE_ID adc10bitId;	Identifies peripheral (PLIB-level) ID

Description

Macro: ADC10BIT Driver Initialization Data

This data type defines the data required to initialize or reinitialize the 10-bit ADC Driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_TOUCH_ADC10BIT_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_HANDLE_INVALID ((DRV_TOUCH_ADC10BIT_HANDLE)(-1))
```

Description

Macro: ADC10BIT Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is returned by DRV_ADC10BIT_RawRead and DRV_ADC10BIT_RawRead functions if the request was not successful.

Remarks

None.

DRV_TOUCH_ADC10BIT_INDEX_0 Macro

ADC10BIT driver index definitions.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_0 0
```

Description

Macro: ADC10BIT Driver Module Index Numbers

These constants provide the 10-bit ADC Driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_ADC10BIT_Initialize and DRV_ADC10BIT_Open functions to identify the driver instance in use.

DRV_TOUCH_ADC10BIT_INDEX_1 Macro

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_1 1
```

Description

This is macro DRV_TOUCH_ADC10BIT_INDEX_1.

DRV_TOUCH_ADC10BIT_INDEX_COUNT Macro

Number of valid ADC10BIT driver indices.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_COUNT 2
```

Description

Macro: ADC10BIT Driver Module Index Count

This constant identifies the number of valid 10-bit ADC Driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_adc10bit.h	10-bit ADC Touch Driver interface definitions
drv_adc10bit_config_template.h	10-bit ADC Touch Driver configuration template.

Description

This section lists the source and header files used by the 10-bit ADC Touch Driver Library.

drv_adc10bit.h

10-bit ADC Touch Driver interface definitions

Enumerations

	Name	Description
	DRV_ADC10BIT_MODULE_ID	This is type DRV_ADC10BIT_MODULE_ID.

Functions

	Name	Description
	DRV_TOUCH_ADC10BIT_CalibrationSet	Loads calibration parameters from Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_Close	Closes an opened instance of the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_Deinitialize	Deinitializes the specified instance of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Initialize	Initializes the 10-bit ADC Driver instance for the specified driver index
	DRV_TOUCH_ADC10BIT_Open	Opens the specified ADC10BIT driver instance and returns a handle to it.
	DRV_TOUCH_ADC10BIT_PositionDetect	None.
	DRV_TOUCH_ADC10BIT_Status	Provides the current status of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_TOUCH_ADC10BIT_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_ADC10BIT_TouchGetRawX	Returns raw x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetRawY	Returns raw y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetX	Returns x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetY	Returns y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchStatus	Returns the status of the current touch input.
	DRV_TOUCH_ADC10BIT_TouchStoreCalibration	Stores calibration parameters into Non-volatile Memory.

Macros

	Name	Description
	DRV_TOUCH_ADC10BIT_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_ADC10BIT_INDEX_0	ADC10BIT driver index definitions.
	DRV_TOUCH_ADC10BIT_INDEX_1	This is macro DRV_TOUCH_ADC10BIT_INDEX_1.
	DRV_TOUCH_ADC10BIT_INDEX_COUNT	Number of valid ADC10BIT driver indices.

Structures

	Name	Description
	_DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.

Types

	Name	Description
	DRV_TOUCH_ADC10BIT_HANDLE	Driver handle.

Description

10-bit ADC Touch Driver Interface Definition

This is a resistive touch screen driver that is using the Microchip Graphics Library. The calibration values are automatically checked (by reading a specific memory location on the non-volatile memory) when initializing the module if the function pointers to the read and write callback functions are initialized. If the read value is invalid calibration will automatically be executed. Otherwise, the calibration values will be loaded and used. The driver assumes that the application side provides the read and write routines to a non-volatile memory. If the callback functions are not initialized, the calibration routine will always be called at start-up to initialize the global calibration values. This driver assumes that the Graphics Library is initialized and will be using the default font of the library.

File Name

drv_adc10bit.h

Company

Microchip Technology Inc.

drv_adc10bit_config_template.h

10-bit ADC Touch Driver configuration template.

Macros

	Name	Description
	DRV_ADC10BIT_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_ADC10BIT_CALIBRATION_INSET	Defines the calibration inset.
	DRV_ADC10BIT_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ADC10BIT_INDEX	ADC10BIT static index selection.
	DRV_ADC10BIT_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_ADC10BIT_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ADC10BIT_SAMPLE_POINTS	Defines the sample points.
	DRV_ADC10BIT_TOUCH_DIAMETER	Defines the touch diameter.

Description

10-bit ADC Touch Driver Configuration Template

This header file contains the build-time configuration selections for the 10-bit ADC Touch Driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_adc10bit_config_template.h

Company

Microchip Technology Inc.

ADC Touch Driver Library

This topic describes the ADC Touch Driver Library.

Introduction

This library provides an interface to manage the ADC Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony ADC Touch Driver provides a high-level interface to the ADC touch device. This driver provides application routines to read the touch input data from the touch screen. . The ADC touch device can notify the availability of touch input data through external interrupt. The ADC Touch Driver allows the application to map a controller pin as an external interrupt pin.

Using the Library

This topic describes the basic architecture of the ADC Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_touch_adc.h`

The interface to the ADC Touch Driver library is defined in the `drv_touch_adc.h` header file. Any C language source (`.c`) file that uses the ADC Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

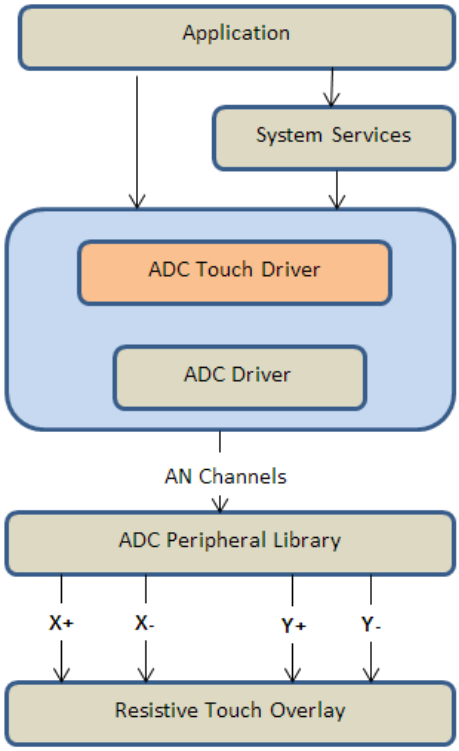
This library provides a low-level abstraction of the ADC Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The ADC Touch Driver has routines to perform the following operations. The driver initialization routines allow the application to initialize the driver. The driver must be initialized before it can be used by application. Once driver is initialized the driver open routine allows retrieving the client handle. Once the touch input is available a touch input read request is sent and input data is retrieved in a buffer. The buffer data is then decoded

to get the x and y coordinate of the touch screen in the form of the number of pixels.

ADC Touch Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADC Touch Driver module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

The library provides interfaces to support system functions, which provide system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

During steady state operation, the DRV_TOUCH_ADC_Tasks is called continuously in System_Tasks to save the current touch position, or -1 if no touch was detected.

At any time, DRV_TOUCH_ADC_TouchGetX and DRV_TOUCH_ADC_TouchGetY are called to retrieve the last touch position. Touch positions are not queued.

Touch samples are configurable, the default is 300. The return integer can have the value between 0-screenWidth and 0-screenHeight.

Initializing the Driver

Before the ADC Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_TOUCH_INIT data structure that is passed to the DRV_TOUCH_ADC_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the ADC Touch Touch Driver.

```
/* The following code shows an example of designing the
 *  DRV_TOUCH_INIT data structure. It also shows how an example
 *  usage of the DRV_TOUCH_ADC_Initialize function.
 */
```

```
DRV_TOUCH_INIT drvTouchInitData;
SYS_MODULE_OBJ objectHandle;

/* Touch Module Id*/
drvTouchInitData.touchId                    = DRV_TOUCH_INDEX_0;

/* I2C Bus driver open */
drvTouchInitData.drvOpen                    = DRV_I2C_Open;

/* Interrupt Source for Touch */
drvTouchInitData.interruptSource            = INT_SOURCE_EXTERNAL_1;

/* Interrupt Pin function mapping */
drvTouchInitData.interruptPort.inputFunction = INPUT_FUNC_INT1;

/* Pin to be mapped as interrupt pin */
drvTouchInitData.interruptPort.inputPin     = INPUT_PIN_RPE8;

/* Analog pin number */
drvTouchInitData.interruptPort.analogPin    = PORTS_ANALOG_PIN_25;

/* Pin Mode of analog pin */
drvTouchInitData.interruptPort.pinMode      = PORTS_PIN_MODE_DIGITAL;

/* Interrupt pin port */
drvTouchInitData.interruptPort.channel      = PORT_CHANNEL_E;

/* Interrupt pin port maskl */
drvTouchInitData.interruptPort.dataMask     = 0x8;

/* Touch screen orientation */
drvTouchInitData.orientation                = DISP_ORIENTATION;

/* Touch screen horizontal resolution */
drvTouchInitData.horizontalResolution           = DISP_HOR_RESOLUTION;

/* Touch screen vertical resolution */
drvTouchInitData.verticalResolution         = DISP_VER_RESOLUTION;

/* Driver initialization */
objectHandle = DRV_TOUCH_ADC_Initialize(DRV_TOUCH_INDEX_0,
                                        (SYS_MODULE_INIT*)drvTouchInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Opening the Driver

To use the ADC Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_ADC_Open function. If successful, the DRV_TOUCH_ADC_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_ADC_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
DRV_HANDLE handle;
handle = DRV_TOUCH_ADC_Open( DRV_TOUCH_ADC_INDEX_0,
DRV_IO_INTENT_EXCLUSIVE );
if( DRV_HANDLE_INVALID == handle )
{
// Unable
}
```

Tasks Routine

To use the ADC Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_ADC_Open function. If successful, the DRV_TOUCH_ADC_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_ADC_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in

other (error) cases as well. The following code shows an example of the driver being opened.

```
DRV_HANDLE handle;
handle = DRV_TOUCH_ADC_Open( DRV_TOUCH_ADC_INDEX_0,
DRV_IO_INTENT_EXCLUSIVE );
if( DRV_HANDLE_INVALID == handle )
{
// Unable
}
```

Configuring the Library

The configuration of the ADC Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the ADC 10-bit Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the ADC Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the ADC Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the ADC Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/touch_adc`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_touch_adc.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/drv_touch_adc.c	Basic ADC Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The ADC Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

This section describes the API functions of the ADC Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

b) Data Types and Constants

Files

This section lists the source and header files used by the ADC Touch Driver Library.

AR1021 Touch Driver Library

This topic describes the AR1021 Touch Driver Library.

Introduction

This library provides a low-level abstraction of the AR1021 Touch Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The AR1021 Touch Driver Library, in conjunction with the Microchip AR1021 Resistive Touch Screen Controller module, allows an application to:

- Calibrate touch points
- Receive touch points

The following application services are provided by the AR1021 Touch Driver Library:

- Configuring the AR1021 controller (TouchThreshold, PenUpDelay, PenStateReportDelaylist, SensitivityFilter, etc.)
- Saving touch points to EEPROM

The operational services are not typically accessible to the application as this portion of the code resides within the Touch System Service Library software layer and is used by the Graphics Library stack services to receive touch point data.

Using the Library

This topic describes the basic architecture of the AR1021 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ar1021.h`

The interface to the AR1021 Touch Driver library is defined in the `drv_ar1021.h` header file. Any C language source (`.c`) file that uses the AR1021 Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the AR1021 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

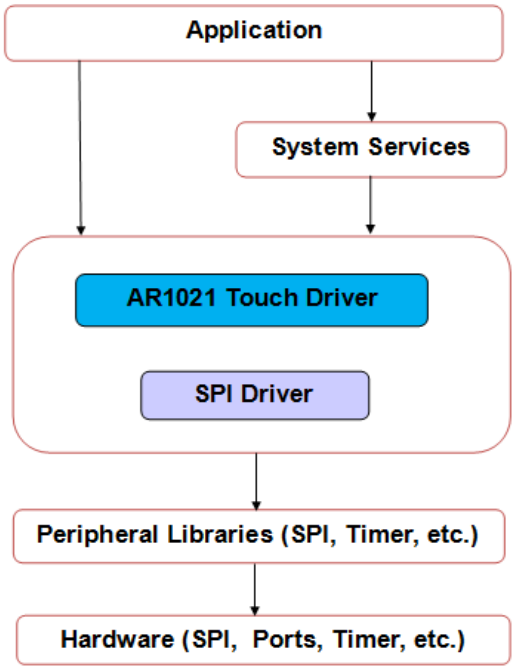
Description

The AR1021 Touch Driver Library provides the following functionality:

- AR1021 library initialization
- AR1021 controller configuration
- AR1021 controller connectivity
- AR1021 polling for pen-down and pen-up touch point events

The abstraction model shown in the following diagram depicts how the AR1021 Touch Driver is positioned in the MPLAB Harmony framework. The AR1021 Touch Driver Library uses the SPI Driver for control and touch data transfers to the AR1021 module.

AR1021 Touch Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AR1021 Touch Driver module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, task, and status functions.
Client Functions	Provides functions to open, close, and calibrate the AR1021 Touch Driver.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, task, touch, and status functions
- Client functions, which open, close, and calibrate the AR1021 Touch Driver

Initializing the Driver

Before the AR1021 Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_TOUCH_INIT data structure that is passed to the DRV_TOUCH_AR1021_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the AR1021 Touch Driver.

Example:

```
/* The following code shows an example of designing the
 *  DRV_TOUCH_INIT data structure. It also shows how an example
 *  usage of the DRV_TOUCH_AR1021_Initialize function.
 */

DRV_TOUCH_INIT drvTouchInitData;
SYS_MODULE_OBJ objectHandle;

/* Driver initialization */
objectHandle = DRV_TOUCH_AR1021_Initialize(DRV_TOUCH_INDEX_0,
```

```
                                        (SYS_MODULE_INIT*)drvTouchInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Opening the Driver

To use the AR1021 Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_AR1021_Open function.

If successful, the DRV_TOUCH_AR1021_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_AR1021_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_AR1021_Open( DRV_TOUCH_AR1021_INDEX_0,
                                DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Tasks Routine

This routine processes the AR1021 Touch Driver commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The AR1021 Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_AR1021_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_AR1021_Tasks ( object );

    // Do other tasks
}
```

Configuring the Library

Macros

	Name	Description
	DRV_AR1021_CALIBRATION_DELAY	Define the calibration delay.
	DRV_AR1021_CALIBRATION_INSET	Define the calibration inset.
	DRV_AR1021_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_AR1021_INDEX	AR1021 static index selection.
	DRV_AR1021_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_AR1021_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_AR1021_SAMPLE_POINTS	Define the sample points.
	DRV_AR1021_TOUCH_DIAMETER	Define the touch diameter.

Description

The configuration of the AR1021 Touch Driver is accomplished through AR1021 Touch Driver selections in the MPLAB Harmony Configurator (MHC). Based on the selections made, a specific AR1021 Touch Driver is established automatically to execute all system configuration, initialization, and steady-state touch acquisitions.

Refer to Volume III: MPLAB Harmony Configurator (MHC) for more details on system configuration. Refer to the Applications Help section for additional information.

DRV_AR1021_CALIBRATION_DELAY Macro

Define the calibration delay.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CALIBRATION_DELAY 300
```

Description

AR1021 Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_AR1021_CALIBRATION_INSET Macro

Define the calibration inset.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CALIBRATION_INSET 25
```

Description

AR1021 Calibration Inset

This macro define the calibration inset.

Remarks

None.

DRV_AR1021_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CLIENTS_NUMBER 1
```

Description

AR1021 Maximum Number of Clients

This definition selected the maximum number of clients that the AR1021 driver can support at run time.

Remarks

None.

DRV_AR1021_INDEX Macro

AR1021 static index selection.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INDEX DRV_AR1021_INDEX_0
```

Description

AR1021 Static Index Selection

AR1021 static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_AR1021_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INSTANCES_NUMBER 1
```

Description

AR1021 hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_AR1021_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INTERRUPT_MODE false
```

Description

AR1021 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of AR1021 operation is desired
- false - Select if polling mode of AR1021 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_AR1021_SAMPLE_POINTS Macro

Define the sample points.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_SAMPLE_POINTS 4
```

Description

AR1021 Sample Points

AR1021 sample points

Remarks

None.

DRV_AR1021_TOUCH_DIAMETER Macro

Define the touch diameter.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_TOUCH_DIAMETER 10
```

Description

AR1021 Touch Diameter

This macro defines the touch diameter

Remarks

None.

Building the Library

This section lists the files that are available in the AR1021 Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the AR1021 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/ar1021`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ar1021.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/drv_ar1021.c`	Basic AR1021 Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AR1021 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_AR1021_Deinitialize	De-initializes the specified instance of the AR1021 driver module.
	DRV_TOUCH_AR1021_FactoryDefaultSet	Set AR1021 controller to factory default configuration settings.
	DRV_TOUCH_AR1021_Initialize	Initializes the AR1021 instance for the specified driver index

	DRV_TOUCH_AR1021_RegisterConfigWrite	Write a value to the given AR1021 configuration register.
	DRV_TOUCH_AR1021_Status	Provides the current status of the AR1021 driver module.
	DRV_TOUCH_AR1021_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_AR1021_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchPenGet	Returns the PEN state of the touch event.
	DRV_TOUCH_AR1021_TouchStatus	Returns the status of the current touch input.

b) Client Functions

	Name	Description
	DRV_TOUCH_AR1021_Calibrate	Calibrate the touch screen
	DRV_TOUCH_AR1021_CalibrationSet	Set calibration with pre-defined points..
	DRV_TOUCH_AR1021_Close	Closes an opened instance of the AR1021 driver
	DRV_TOUCH_AR1021_Open	Opens the specified AR1021 driver instance and returns a handle to it. **Implementation:** Dynamic

c) Data Types and Constants

	Name	Description
	DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK	Defines the callback functions required to inform the user of touch and release targets.
	DRV_TOUCH_AR1021_HANDLE	Touch screen controller AR1021 driver handle.
	DRV_TOUCH_AR1021_MODULE_ID	This is type DRV_TOUCH_AR1021_MODULE_ID.
	DRV_TOUCH_AR1021_TASK_STATE	Enumeration defining AR1021 touch controller driver task state.
	DRV_TOUCH_AR1021_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_AR1021_INDEX_0	AR1021 driver index definitions.
	DRV_TOUCH_AR1021_INDEX_COUNT	Number of valid AR1021 driver indices.

Description

This section describes the API functions of the AR1021 Touch Driver Library.

a) System Functions

DRV_TOUCH_AR1021_Deinitialize Function

De-initializes the specified instance of the AR1021 driver module.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

De-initializes the specified instance of the AR1021 driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_AR1021_Status operation. The system has to use DRV_TOUCH_AR1021_Status to find out when the module is in the ready state.

Preconditions

Function DRV_TOUCH_AR1021_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_TOUCH_AR1021_Initialize
SYS_STATUS          status;

DRV_TOUCH_AR1021_Deinitialize ( object );

status = DRV_TOUCH_AR1021_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_AR1021_Initialize

Function

void DRV_TOUCH_AR1021_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_AR1021_FactoryDefaultSet Function

Set AR1021 controller to factory default configuration settings.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_FactoryDefaultSet();
```

Returns

None

Description

This function returns the AR1021 to operate on factory default configuration settings.

Remarks

A power cycle is required to run on the default settings.

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_FactoryDefaultSet ( void );
```

Function

void DRV_TOUCH_AR1021_FactoryDefaultSet(void)

DRV_TOUCH_AR1021_Initialize Function

Initializes the AR1021 instance for the specified driver index

File

drv_ar1021.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_AR1021_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AR1021 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the AR1021 module ID. For example, driver instance 0 can be assigned to AR10212. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_AR1021_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other AR1021 routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_AR1021_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_INIT      drvAr1021InitData;
SYS_MODULE_OBJ      objectHandle;

objectHandle = DRV_TOUCH_AR1021_Initialize(DRV_TOUCH_AR1021_INDEX_1, (SYS_MODULE_INIT*)drvAr1021InitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the AR1021 id. The hardware AR1021 id is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_AR1021_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_TOUCH_AR1021_RegisterConfigWrite Function

Write a value to the given AR1021 configuration register.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value);
```

Returns

None

Description

This function set a value to the given AR1021 configuration register.

Remarks

none

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value);
```

Function

void DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value)

DRV_TOUCH_AR1021_Status Function

Provides the current status of the AR1021 driver module.

File

drv_ar1021.h

C

```
SYS_STATUS DRV_TOUCH_AR1021_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the AR1021 driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_TOUCH_AR1021_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_AR1021_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_AR1021_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_AR1021_Initialize

Function

SYS_STATUS DRV_TOUCH_AR1021_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_AR1021_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_AR1021_Initialize

while( true )
{
    DRV_TOUCH_AR1021_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_AR1021_Initialize)

Function

void DRV_TOUCH_AR1021_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_AR1021_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_AR1021_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_AR1021_TouchGetX Function

Returns the x coordinate of touch input.

Implementation: Dynamic

File

drv_ar1021.h

C

```
short DRV_TOUCH_AR1021_TouchGetX(uint8_t touchNumber);
```

Returns

It returns the x coordinate of the touch input in terms of number of pixels.

Description

It returns the x coordinate in form of number of pixels for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_AR1021_TouchGetX(uint8 touchNumber)

DRV_TOUCH_AR1021_TouchGetY Function

Returns the y coordinate of touch input.

Implementation: Dynamic

File

drv_ar1021.h

C

```
short DRV_TOUCH_AR1021_TouchGetY(uint8_t touchNumber);
```

Returns

It returns the y coordinate of the touch input in terms of number of pixels.

Description

It returns the y coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_AR1021_TouchGetY(uint8 touchNumber)

DRV_TOUCH_AR1021_TouchPenGet Function

Returns the PEN state of the touch event.

File

drv_ar1021.h

C

```
DRV_TOUCH_PEN_STATE DRV_TOUCH_AR1021_TouchPenGet(uint8_t touchNumber);
```

Returns

It returns DRV_TOUCH_PEN_STATE

Description

It returns the PEN state of the last touch event corresponding to the x and y position.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

DRV_TOUCH_PEN_STATE DRV_TOUCH_AR1021_TouchPenGet(uint8_t touchNumber)

DRV_TOUCH_AR1021_TouchStatus Function

Returns the status of the current touch input.

File

drv_ar1021.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_AR1021_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_AR1021_TouchStatus(const SYS_MODULE_INDEX index)

b) Client Functions

DRV_TOUCH_AR1021_Calibrate Function

Calibrate the touch screen

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Calibrate(const DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK * prompt);
```

Returns

None

Description

This function display calibration points on the display to enable calibration.

Remarks

None

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_Calibrate ( handle );
```

Function

void DRV_TOUCH_AR1021_Calibrate ((const DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK * prompt))

DRV_TOUCH_AR1021_CalibrationSet Function

Set calibration with pre-defined points..

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_CalibrationSet(DRV_TOUCH_SAMPLE_POINTS * samplePoints);
```

Returns

None

Description

This function allows for the setting of pre-loaded calibration points.

Remarks

None

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_CalibrationSet ( void );
```

Function

void DRV_TOUCH_AR1021_CalibrationSet(void)

DRV_TOUCH_AR1021_Close Function

Closes an opened instance of the AR1021 driver

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the AR1021 driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_AR1021_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

DRV_TOUCH_AR1021_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_AR1021_Open

DRV_TOUCH_AR1021_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_AR1021_Close (DRV_HANDLE handle)

DRV_TOUCH_AR1021_Open Function

Opens the specified AR1021 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ar1021.h

C

```
DRV_HANDLE DRV_TOUCH_AR1021_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_AR1021_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified AR1021 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the AR1021 device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_TOUCH_AR1021_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_TOUCH_AR1021_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_AR1021_Open( DRV_TOUCH_AR1021_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_AR1021_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_TOUCH_AR1021_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

c) Data Types and Constants

DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK Structure

Defines the callback functions required to inform the user of touch and release targets.

File

drv_ar1021.h

C

```
typedef struct {
  void (* firstPromptCallback)(void);
  void (* secondPromptCallback)(void);
  void (* thirdPromptCallback)(void);
  void (* fourthPromptCallback)(void);
  void (* completeCallback)(void);
} DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK;
```

Members

Members	Description
void (* firstPromptCallback)(void);	first calibration target
void (* secondPromptCallback)(void);	second calibration target
void (* thirdPromptCallback)(void);	third calibration target
void (* fourthPromptCallback)(void);	fourth calibration target
void (* completeCallback)(void);	complete calibration

Description

TOUCH Driver Calibration Initialization Data

This data type defines the callback function pointers required to inform of touch and release targets. The driver will invoke each callback in sequential order. The host code can display graphic and/or textual content to direct the user when a where on the LCD display to touch and release.

Remarks

None.

DRV_TOUCH_AR1021_HANDLE Type

Touch screen controller AR1021 driver handle.

File

drv_ar1021.h

C

```
typedef uintptr_t DRV_TOUCH_AR1021_HANDLE;
```

Description

AR1021 Driver Handle

Touch controller AR1021 driver handle is a handle for the driver client object. Each driver with successful open call will return a new handle to the client object.

Remarks

None.

DRV_TOUCH_AR1021_MODULE_ID Enumeration

File

drv_ar1021.h

C

```
typedef enum {
  AR1021_ID_1 = 0,
  AR1021_NUMBER_OF_MODULES
} DRV_TOUCH_AR1021_MODULE_ID;
```

Description

This is type DRV_TOUCH_AR1021_MODULE_ID.

DRV_TOUCH_AR1021_TASK_STATE Enumeration

Enumeration defining AR1021 touch controller driver task state.

File

drv_ar1021.h

C

```
typedef enum {
  DRV_TOUCH_AR1021_TASK_STATE_INIT = 0,
  DRV_TOUCH_AR1021_TASK_STATE_DONE
} DRV_TOUCH_AR1021_TASK_STATE;
```

Members

Members	Description
DRV_TOUCH_AR1021_TASK_STATE_INIT = 0	Task initialize state
DRV_TOUCH_AR1021_TASK_STATE_DONE	Task complete state

Description

AR1021 Touch Controller Driver Task State

This enumeration defines the AR1021 touch controller driver task state. The task state helps to synchronize the operations of initialization the the task, adding the read input task to the task queue once the touch controller notifies the available touch input and a decoding the touch input received.

Remarks

None.

DRV_TOUCH_AR1021_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_HANDLE_INVALID ((DRV_TOUCH_AR1021_HANDLE)(-1))
```

Description

AR1021 Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_TOUCH_AR1021_Open() and DRV_AR1021_Close() functions if the request was not successful.

Remarks

None.

DRV_TOUCH_AR1021_INDEX_0 Macro

AR1021 driver index definitions.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_INDEX_0 0
```

Description

AR1021 Driver Module Index Numbers

These constants provide the AR1021 driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AR1021_Initialize and

DRV_AR1021_Open functions to identify the driver instance in use.

DRV_TOUCH_AR1021_INDEX_COUNT Macro

Number of valid AR1021 driver indices.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_INDEX_COUNT 1
```

Description

AR1021 Driver Module Index Count

This constant identifies the number of valid AR1021 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_ar1021.h	Touch controller AR1021 driver implementation.

Description

This section lists the source and header files used by the AR1021 Touch Driver Library.

drv_ar1021.h

Touch controller AR1021 driver implementation.

Enumerations

	Name	Description
	DRV_TOUCH_AR1021_MODULE_ID	This is type DRV_TOUCH_AR1021_MODULE_ID.
	DRV_TOUCH_AR1021_TASK_STATE	Enumeration defining AR1021 touch controller driver task state.

Functions

	Name	Description
	DRV_TOUCH_AR1021_Calibrate	Calibrate the touch screen
	DRV_TOUCH_AR1021_CalibrationSet	Set calibration with pre-defined points..
	DRV_TOUCH_AR1021_Close	Closes an opened instance of the AR1021 driver
	DRV_TOUCH_AR1021_Deinitialize	De-initializes the specified instance of the AR1021 driver module.
	DRV_TOUCH_AR1021_FactoryDefaultSet	Set AR1021 controller to factory default configuration settings.
	DRV_TOUCH_AR1021_Initialize	Initializes the AR1021 instance for the specified driver index
	DRV_TOUCH_AR1021_Open	Opens the specified AR1021 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_RegisterConfigWrite	Write a value to the given AR1021 configuration register.
	DRV_TOUCH_AR1021_Status	Provides the current status of the AR1021 driver module.
	DRV_TOUCH_AR1021_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_AR1021_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchPenGet	Returns the PEN state of the touch event.

	DRV_TOUCH_AR1021_TouchStatus	Returns the status of the current touch input.

Macros

	Name	Description
	DRV_TOUCH_AR1021_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_AR1021_INDEX_0	AR1021 driver index definitions.
	DRV_TOUCH_AR1021_INDEX_COUNT	Number of valid AR1021 driver indices.

Structures

	Name	Description
	DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK	Defines the callback functions required to inform the user of touch and release targets.

Types

	Name	Description
	DRV_TOUCH_AR1021_HANDLE	Touch screen controller AR1021 driver handle.

Description

Touch controller AR1021 driver file

This file consist of touch controller AR1021 driver interfaces. It implements the driver interfaces which read the touch input data from AR1021 through SPI bus.

File Name

drv_ar1021.c

MTCH6301 Touch Driver Library

This topic describes the MTCH6301 Touch Driver Library.

Introduction

This library provides an interface to manage the MTCH6301 Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony MTCH6301 Touch Driver provides a high-level interface to the MTCH6301 touch controller device. This driver provides application routines to read the touch input data from the touch screen. The MTCH6301 device can notify the availability of touch input data through external interrupt. The MTCH6301 driver allows the application to map a controller pin as an external interrupt pin.

Currently, the MTCH6301 Touch Driver only supports non-gestural single-fingered touch input.

Using the Library

This topic describes the basic architecture of the MTCH6301 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_mtch6301.h`

The interface to the MTCH6301 Touch Driver library is defined in the `drv_mtch6301.h` header file. Any C language source (`.c`) file that uses the MTCH6301 Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MTCH6301 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

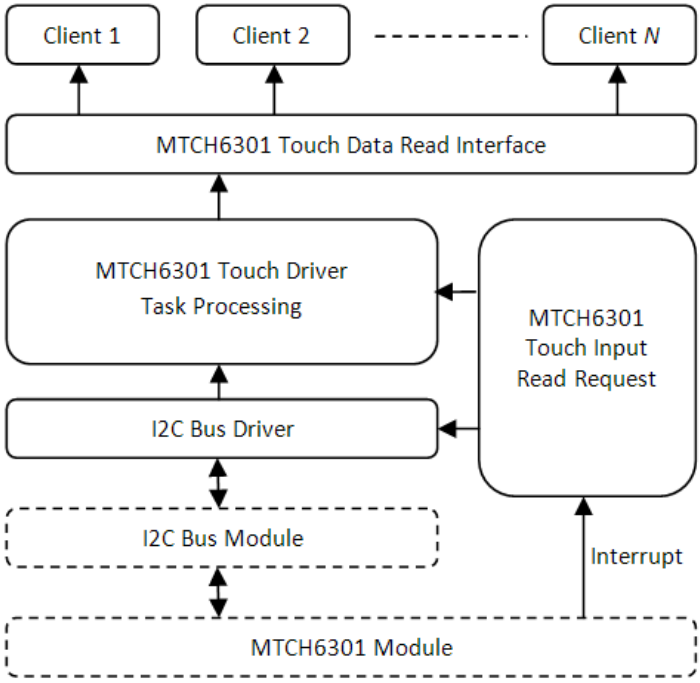
The MTCH6301 Touch Driver has routines to perform the following operations:

- Sending read request
- Reading the touch input data

- Access to the touch input data

The driver initialization routines allow the application to initialize the driver. The driver must be initialized before it can be used by application. Once the driver is initialized the driver open routine allows to retrieve the client handle. Once the touch input is available a touch input read request is sent and input data is retrieved in a buffer. The buffer data is then decoded to get the x and y coordinate of the touch screen in the form of the number of pixels.

MTCH6301 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MTCH6301 Touch Driver.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, open, close, task, and status functions.
- Read Request function, which provides Touch input data read request function
- Read Touch Input function, which provides functions retrieving updated Touch input in the form x and y coordinates.

Initializing the Driver

Before the MTCH6301 Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_TOUCH_INIT data structure that is passed to the DRV_TOUCH_MTCH6301_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the MTCH6301 Touch Driver.

Example:

```
/* The following code shows an example of designing the
 *  DRV_TOUCH_INIT data structure. It also shows how an example
 *  usage of the DRV_TOUCH_MTCH6301_Initialize function.
 */
```

```
DRV_TOUCH_INIT drvTouchInitData;
SYS_MODULE_OBJ objectHandle;

/* Touch Module Id*/
drvTouchInitData.touchId                    = DRV_TOUCH_INDEX_0;

/* I2C Bus driver open */
drvTouchInitData.drvOpen                    = DRV_I2C_Open;

/* Interrupt Source for Touch */
drvTouchInitData.interruptSource            = INT_SOURCE_EXTERNAL_1;

/* Interrupt Pin function mapping */
drvTouchInitData.interruptPort.inputFunction = INPUT_FUNC_INT1;

/* Pin to be mapped as interrupt pin */
drvTouchInitData.interruptPort.inputPin     = INPUT_PIN_RPE8;

/* Analog pin number */
drvTouchInitData.interruptPort.analogPin    = PORTS_ANALOG_PIN_25;

/* Pin Mode of analog pin */
drvTouchInitData.interruptPort.pinMode      = PORTS_PIN_MODE_DIGITAL;

/* Interrupt pin port */
drvTouchInitData.interruptPort.channel      = PORT_CHANNEL_E;

/* Interrupt pin port maskl */
drvTouchInitData.interruptPort.dataMask     = 0x8;

/* Touch screen orientation */
drvTouchInitData.orientation                = DISP_ORIENTATION;

/* Touch screen horizontal resolution */
drvTouchInitData.horizontalResolution           = DISP_HOR_RESOLUTION;

/* Touch screen vertical resolution */
drvTouchInitData.verticalResolution         = DISP_VER_RESOLUTION;

/* Driver initialization */
objectHandle = DRV_TOUCH_MTCH6301_Initialize(DRV_TOUCH_INDEX_0,
                                             (SYS_MODULE_INIT*)drvTouchInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Opening the Driver

To use the MTCH6301 Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_MTCH6301_Open function.

If successful, the DRV_TOUCH_MTCH6301_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_MTCH6301_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
    DRV_HANDLE  handle;

    handle = DRV_TOUCH_MTCH6301_Open( DRV_TOUCH_MTCH6301_INDEX_0,
                                      DRV_IO_INTENT_EXCLUSIVE );

    if( DRV_HANDLE_INVALID == handle )
    {
        // Unable to open the driver
    }
```

Touch Input Read Request

To read the touch input from the MTCH6301 touch controller device, a read request must be registered. This is done by calling the DRV_TOUCH_MTCH6301_ReadRequest. If successful it registers a buffer read request to the I2C command queue. It also adds a input decode command to the MTCH6301 command queue once the I2C returns with touch input data. It can return error if the driver instance object is invalid or the MTCH6301 command queue is full. The read request is to be called from the MTCH6301 ISR. This ISR is triggered once the touch input is available. The following code shows an example of a MTCH6301 read request registration:

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMtch6301(void)
{
    DRV_TOUCH_MTCH6301_ReadRequest ( object );

    // Do other tasks
}
```

Tasks Routine

This routine processes the MTCH6301 commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The MTCH6301 Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_MTCH6301_Tasks ( object );

    // Do other tasks
}
```

Configuring the Library

Macros

	Name	Description
	DRV_MTCH6301_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_MTCH6301_CALIBRATION_INSET	Defines the calibration inset.
	DRV_MTCH6301_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_MTCH6301_INDEX	MTCH6301 static index selection.
	DRV_MTCH6301_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_MTCH6301_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_MTCH6301_SAMPLE_POINTS	Define the sample points.
	DRV_MTCH6301_TOUCH_DIAMETER	Defines the touch diameter.

Description

The configuration of the MTCH6301 Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the MTCH6301 Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the MTCH6301 Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_MTCH6301_CALIBRATION_DELAY Macro

Defines the calibration delay.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CALIBRATION_DELAY 300
```

Description

MTCH6301 Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_MTCH6301_CALIBRATION_INSET Macro

Defines the calibration inset.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CALIBRATION_INSET 25
```

Description

MTCH6301 Calibration Inset

This macro defines the calibration inset.

Remarks

None.

DRV_MTCH6301_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CLIENTS_NUMBER 1
```

Description

MTCH6301 maximum number of clients

This macro selects the maximum number of clients.

This definition selected the maximum number of clients that the MTCH6301 driver can support at run time.

Remarks

None.

DRV_MTCH6301_INDEX Macro

MTCH6301 static index selection.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INDEX DRV_MTCH6301_INDEX_0
```

Description

MTCH6301 Static Index Selection

This macro specifies the static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_MTCH6301_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INSTANCES_NUMBER 1
```

Description

MTCH6301 hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_MTCH6301_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INTERRUPT_MODE false
```

Description

MTCH6301 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of MTCH6301 operation is desired
- false - Select if polling mode of MTCH6301 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_MTCH6301_SAMPLE_POINTS Macro

Define the sample points.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_SAMPLE_POINTS 4
```

Description

MTCH6301 Sample Points

MTCH6301 sample points

Remarks

None.

DRV_MTCH6301_TOUCH_DIAMETER Macro

Defines the touch diameter.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_TOUCH_DIAMETER 10
```

Description

MTCH6301 Touch Diameter

This macro defines the touch diameter

Remarks

None.

Building the Library

This section lists the files that are available in the MTCH6301 Touch Driver Library.

Description

This section list the files that are available in the \src folder of the MTCH6301 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/mtch6301`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_mtch6301.h	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/drv_mtch6301.c	Basic MTCH6301 Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The MTCH6301 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_MTCH6301_Close	Closes an opened instance of the MTCH6301 driver. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Deinitialize	Deinitializes the specified instance of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Initialize	Initializes the MTCH6301 instance for the specified driver index. **Implementation:** Dynamic

	DRV_TOUCH_MTCH6301_Open	Opens the specified MTCH6301 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Status	Provides the current status of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_MTCH6301_TouchStatus	Returns the status of the current touch input.

b) Data Types and Constants

	Name	Description
	_DRV_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_HANDLE	Touch screen controller MTCH6301 driver handle.
	DRV_TOUCH_MTCH6301_MODULE_ID	Number of valid MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE	I2C Frame size for reading MTCH6301 touch input.
	DRV_TOUCH_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_INDEX_0	MTCH6301 driver index definitions.
	DRV_TOUCH_MTCH6301_INDEX_1	This is macro DRV_TOUCH_MTCH6301_INDEX_1.
	DRV_TOUCH_MTCH6301_INDEX_COUNT	Number of valid Touch controller MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_OBJECT	Defines the data structure maintaining MTCH6301 driver instance object.
	DRV_TOUCH_MTCH6301_TASK_QUEUE	Defines the MTCH6301 Touch Controller driver task data structure.
	DRV_TOUCH_MTCH6301_TASK_STATE	Enumeration defining MTCH6301 touch controller driver task state.
	DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID	MTCH6301 input read, I2C address from where master reads touch input data.
	DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID	MTCH6301 command register write, I2C address where master sends the commands.

Description

This section describes the API functions of the MTCH6301 Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_TOUCH_MTCH6301_Close Function

Closes an opened instance of the MTCH6301 driver.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MTCH6301 driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_MTCH6301_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

DRV_TOUCH_MTCH6301_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_MTCH6301_Open

DRV_TOUCH_MTCH6301_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_MTCH6301_Close (DRV_HANDLE handle)

DRV_TOUCH_MTCH6301_Deinitialize Function

Deinitializes the specified instance of the MTCH6301 driver module.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the MTCH6301 driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_MTCH6301_Status operation. The system has to use DRV_TOUCH_MTCH6301_Status to determine when the module is in the ready state.

Preconditions

Function DRV_TOUCH_MTCH6301_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Parameter: object - Driver object handle, returned from DRV_TOUCH_MTCH6301_Initialize

Example

```
SYS_MODULE_OBJ      object;    //Returned from DRV_TOUCH_MTCH6301_Initialize
SYS_STATUS          status;

DRV_TOUCH_MTCH6301_Deinitialize ( object );

status = DRV_TOUCH_MTCH6301_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_TOUCH_MTCH6301_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_Initialize Function

Initializes the MTCH6301 instance for the specified driver index.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_MTCH6301_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MTCH6301 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the MTCH6301 module ID. For example, driver instance 0 can be assigned to MTCH63012. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_MTCH6301_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other MTCH6301 routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_MTCH6301_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_MTCH6301_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the MTCH6301 initialization structure
// Touch Module Id
init.touchId                      = DRV_TOUCH_INDEX_0;

// I2C Bus driver open
init.drvOpen                      = DRV_I2C_Open;

// Interrupt Source for Touch
init.interruptSource              = INT_SOURCE_EXTERNAL_1;

// Interrupt Pin function mapping
init.interruptPort.inputFunction  = INPUT_FUNC_INT1;

// Pin to be mapped as interrupt pin
init.interruptPort.inputPin       = INPUT_PIN_RPE8;

// Analog pin number
init.interruptPort.analogPin      = PORTS_ANALOG_PIN_25;

// Pin Mode of analog pin
init.interruptPort.pinMode        = PORTS_PIN_MODE_DIGITAL;

// Interrupt pin port
init.interruptPort.channel        = PORT_CHANNEL_E;

// Interrupt pin port maskl
init.interruptPort.dataMask       = 0x8;
```

```
// Touch screen orientation
init.orientation                = DISP_ORIENTATION;

// Touch screen horizontal resolution
init.horizontalResolution       = DISP_HOR_RESOLUTION;

// Touch screen vertical resolution
init.verticalResolution         = DISP_VER_RESOLUTION;


objectHandle = DRV_TOUCH_MTCH6301_Initialize(DRV_TOUCH_INDEX_0,
                                             (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the MTCH6301 ID. The hardware MTCH6301 ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_MTCH6301_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_TOUCH_MTCH6301_Open Function

Opens the specified MTCH6301 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
DRV_HANDLE DRV_TOUCH_MTCH6301_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_MTCH6301_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified MTCH6301 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the MTCH6301 device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_TOUCH_MTCH6301_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;
```

```
handle = DRV_TOUCH_MTCH6301_Open( DRV_TOUCH_MTCH6301_INDEX_0,
                                  DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_MTCH6301_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_TOUCH_MTCH6301_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_TOUCH_MTCH6301_Status Function

Provides the current status of the MTCH6301 driver module.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
SYS_STATUS DRV_TOUCH_MTCH6301_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system-level operation and cannot start another

Description

This function provides the current status of the MTCH6301 driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;  // Returned from DRV_TOUCH_MTCH6301_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_MTCH6301_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_MTCH6301_Initialize

Function

SYS_STATUS DRV_TOUCH_MTCH6301_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function. This routine decodes the touch input data available in drvI2CReadFrameData.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_MTCH6301_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_MTCH6301_Initialize)

Function

void DRV_TOUCH_MTCH6301_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_MTCH6301_ReadRequest Function

Sends a read request to I2C bus driver and adds the read task to queue.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_ReadRequest(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to send a touch input read request to the I2C bus driver and adding the input read decode task to the queue. It is always called from MTCH6301 interrupt ISR routine.

Remarks

This function is normally not called directly by an application. It is called by the MTCH6301 ISR routine.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMtch6301(void)
{
    DRV_TOUCH_MTCH6301_ReadRequest ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_MTCH6301_Initialize)

Function

void DRV_TOUCH_MTCH6301_ReadRequest(SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_TouchGetX Function

Returns the x coordinate of touch input.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
short DRV_TOUCH_MTCH6301_TouchGetX(uint8_t touchNumber);
```

Returns

It returns the x coordinate of the touch input in terms of number of pixels.

Description

It returns the x coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_MTCH6301_TouchGetX(uint8 touchNumber)

DRV_TOUCH_MTCH6301_TouchGetY Function

Returns the y coordinate of touch input.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
short DRV_TOUCH_MTCH6301_TouchGetY(uint8_t touchNumber);
```

Returns

It returns the y coordinate of the touch input in terms of number of pixels.

Description

It returns the y coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_MTCH6301_TouchGetY(uint8 touchNumber)

DRV_TOUCH_MTCH6301_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_MTCH6301_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_MTCH6301_TouchStatus Function

Returns the status of the current touch input.

File

drv_mtch6301.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_MTCH6301_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_MTCH6301_TouchStatus(const SYS_MODULE_INDEX index)

b) Data Types and Constants

DRV_TOUCH_MTCH6301_HANDLE Type

Touch screen controller MTCH6301 driver handle.

File

drv_mtch6301.h

C

```
typedef uintptr_t DRV_TOUCH_MTCH6301_HANDLE;
```

Description

MTCH6301 Driver Handle

Touch controller MTCH6301 driver handle is a handle for the driver client object. Each driver with succesful open call will return a new handle to the client object.

Remarks

None.

DRV_TOUCH_MTCH6301_MODULE_ID Enumeration

Number of valid MTCH6301 driver indices.

File

drv_mtch6301.h

C

```
typedef enum {
  MTCH6301_ID_1 = 0,
  MTCH6301_NUMBER_OF_MODULES
} DRV_TOUCH_MTCH6301_MODULE_ID;
```

Description

MTCH6301 Driver Module Index Count

This constant identifies the number of valid MTCH6301 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_TOUCH_MTCH6301_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_HANDLE_INVALID ((DRV_TOUCH_MTCH6301_HANDLE)(-1))
```

Description

MTCH6301 Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_TOUCH_MTCH6301_Open() and DRV_MTCH6301_Close() functions if the request was not successful.

Remarks

None.

DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE Macro

I2C Frame size for reading MTCH6301 touch input.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE 7
```

Description

MTCH6301 Driver Module I2C Frame Size

This constant identifies the size of I2C frame required to read from MTCH6301 touch controller. MTCH6301 notifies the availability of input data through interrupt pin.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_TOUCH_MTCH6301_CLIENT_OBJECT Structure

MTCH6301 Driver client object maintaining client data.

File

drv_mtch6301.h

C

```
typedef struct _DRV_MTCH6301_CLIENT_OBJECT {
  DRV_TOUCH_MTCH6301_OBJECT* driverObject;
  DRV_IO_INTENT intent;
  struct DRV_TOUCH_MTCH6301_CLIENT_OBJECT* pNext;
} DRV_TOUCH_MTCH6301_CLIENT_OBJECT;
```

Members

Members	Description
DRV_TOUCH_MTCH6301_OBJECT* driverObject;	Driver Object associated with the client
DRV_IO_INTENT intent;	The intent with which the client was opened
struct DRV_TOUCH_MTCH6301_CLIENT_OBJECT* pNext;	Next driver client object

Description

MTCH6301 Driver client object

This defines the object required for the maintenance of the software clients instance. This object exists once per client instance.

Remarks

None.

DRV_TOUCH_MTCH6301_INDEX_0 Macro

MTCH6301 driver index definitions.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_0 0
```

Description

MTCH6301 Driver Module Index Numbers

These constants provide the MTCH6301 driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_MTCH6301_Initialize and DRV_MTCH6301_Open functions to identify the driver instance in use.

DRV_TOUCH_MTCH6301_INDEX_1 Macro

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_1 1
```

Description

This is macro DRV_TOUCH_MTCH6301_INDEX_1.

DRV_TOUCH_MTCH6301_INDEX_COUNT Macro

Number of valid Touch controller MTCH6301 driver indices.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_COUNT 2
```

Description

MTCH6301 Driver Module Index Count

This constant identifies the number of valid Touch Controller MTCH6301 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_TOUCH_MTCH6301_OBJECT Structure

Defines the data structure maintaining MTCH6301 driver instance object.

File

drv_mtch6301.h

C

```
typedef struct {
  SYS_STATUS status;
  int touchId;
  SYS_MODULE_INDEX drvIndex;
  bool inUse;
  bool isExclusive;
  uint8_t numClients;
  INT_SOURCE interruptSource;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
  DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  int32_t readRequest;
  DRV_TOUCH_MTCH6301_TASK_QUEUE* taskQueue;
  DRV_HANDLE drvI2CHandle;
  DRV_TOUCH_POSITION_STATUS touchStatus;
} DRV_TOUCH_MTCH6301_OBJECT;
```

Members

Members	Description
SYS_STATUS status;	The status of the driver
int touchId;	The peripheral Id associated with the object
SYS_MODULE_INDEX drvIndex;	Save the index of the driver. Important to know this as we are using reference based accessing
bool inUse;	Flag to indicate instance in use
bool isExclusive;	Flag to indicate module used in exclusive access mode

uint8_t numClients;	Number of clients possible with the hardware instance
INT_SOURCE interruptSource;	Touch input interrupt source
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels
uint16_t verticalResolution;	Vertical Resolution of the displayed orientaion in Pixels
DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);	Callback for I2C Driver Open call
int32_t readRequest;	Touch Input read request counter
DRV_TOUCH_MTCH6301_TASK_QUEUE* taskQueue;	Head of the task queue
DRV_HANDLE drvI2CHandle;	I2C bus driver handle
DRV_TOUCH_POSITION_STATUS touchStatus;	Touch status

Description

MTCH6301 Driver Instance Object.

This data structure maintains the MTCH6301 driver instance object. The object exists once per hardware instance.

Remarks

None.

DRV_TOUCH_MTCH6301_TASK_QUEUE Structure

Defines the MTCH6301 Touch Controller driver task data structure.

File

drv_mtch6301.h

C

```
typedef struct {
  bool inUse;
  DRV_TOUCH_MTCH6301_TASK_STATE taskState;
  DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;
  uint8_t drvI2CReadFrameData[DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE];
} DRV_TOUCH_MTCH6301_TASK_QUEUE;
```

Members

Members	Description
bool inUse;	Flag denoting the allocation of task
DRV_TOUCH_MTCH6301_TASK_STATE taskState;	Enum maintaining the task state
DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;	I2C Buffer handle
uint8_t drvI2CReadFrameData[DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE];	Response to Read Touch Input Command • Response = { MTCH6301 Read Address, • Input Data Size, • Touch Id, Pen status, • Touch X coordinate (0 to 6), • Touch X coordinate (7 to 11), • Touch Y coordinate (0 to 6), • Touch Y coordinate (7 to 11) }

Description

MTCH6301 Touch Controller driver task data structure.

This data type defines the data structure maintaing task context in the task queue. The inUse flag denotes the task context allocation for a task. The enum variable taskState maintains the current task state. The I2C buffer handle drvI2CReadBufferHandle maintains the I2C driver buffer handle returned by the I2C driver read request. The byte array variable drvI2CReadFrameData maintains the I2C frame data sent by MTCH6301 after a successful read request.

Remarks

None.

DRV_TOUCH_MTCH6301_TASK_STATE Enumeration

Enumeration defining MTCH6301 touch controller driver task state.

File

drv_mtch6301.h

C

```
typedef enum {
  DRV_TOUCH_MTCH6301_TASK_STATE_INIT = 0,
  DRV_TOUCH_MTCH6301_TASK_STATE_READ_INPUT,
  DRV_TOUCH_MTCH6301_TASK_STATE_DECODE_INPUT,
  DRV_TOUCH_MTCH6301_TASK_STATE_DONE
} DRV_TOUCH_MTCH6301_TASK_STATE;
```

Members

Members	Description
DRV_TOUCH_MTCH6301_TASK_STATE_INIT = 0	Task initialize state
DRV_TOUCH_MTCH6301_TASK_STATE_READ_INPUT	Task read touch input request state
DRV_TOUCH_MTCH6301_TASK_STATE_DECODE_INPUT	Task touch input decode state
DRV_TOUCH_MTCH6301_TASK_STATE_DONE	Task complete state

Description

MTCH6301 Touch Controller Driver Task State

This enumeration defines the MTCH6301 touch controller driver task state. The task state helps to synchronize the operations of initialization the the task, adding the read input task to the task queue once the touch controller notifies the available touch input and a decoding the touch input received.

Remarks

None.

DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID Macro

MTCH6301 input read, I2C address from where master reads touch input data.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID 0x4B
```

Description

MTCH6301 Driver Module Master Input Read I2C address

This constant defines the MTCH6301 touch input read I2C address. This address is used as I2C address to read Touch input from MTCH6301 Touch controller.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific data sheets.

DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID Macro

MTCH6301 command register write, I2C address where master sends the commands.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID 0x4A
```

Description

MTCH6301 Driver Module Master Command Write I2C Address

This constant defines the MTCH6301 command register I2C write address. This address is used as I2C address to write commands into MTCH6301 Touch controller register.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

Files

Files

Name	Description
drv_mtch6301.h	Touch controller MTCH6301 Driver interface header file.
drv_mtch6301_config_template.h	MTCH6301 Touch Driver configuration template.

Description

This section lists the source and header files used by the MTCH6301 Touch Driver Library.

drv_mtch6301.h

Touch controller MTCH6301 Driver interface header file.

Enumerations

	Name	Description
	DRV_TOUCH_MTCH6301_MODULE_ID	Number of valid MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_TASK_STATE	Enumeration defining MTCH6301 touch controller driver task state.

Functions

	Name	Description
	DRV_TOUCH_MTCH6301_Close	Closes an opened instance of the MTCH6301 driver. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Deinitialize	Deinitializes the specified instance of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Initialize	Initializes the MTCH6301 instance for the specified driver index. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Open	Opens the specified MTCH6301 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Status	Provides the current status of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_MTCH6301_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchStatus	Returns the status of the current touch input.

Macros

	Name	Description
	DRV_TOUCH_MTCH6301_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID	MTCH6301 input read, I2C address from where master reads touch input data.
	DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID	MTCH6301 command register write, I2C address where master sends the commands.

	DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE	I2C Frame size for reading MTCH6301 touch input.
	DRV_TOUCH_MTCH6301_INDEX_0	MTCH6301 driver index definitions.
	DRV_TOUCH_MTCH6301_INDEX_1	This is macro DRV_TOUCH_MTCH6301_INDEX_1.
	DRV_TOUCH_MTCH6301_INDEX_COUNT	Number of valid Touch controller MTCH6301 driver indices.

Structures

	Name	Description
	_DRV_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_OBJECT	Defines the data structure maintaining MTCH6301 driver instance object.
	DRV_TOUCH_MTCH6301_TASK_QUEUE	Defines the MTCH6301 Touch Controller driver task data structure.

Types

	Name	Description
	DRV_TOUCH_MTCH6301_HANDLE	Touch screen controller MTCH6301 driver handle.

Description

Touch Controller MTCH6301 Driver Interface File

This header file describes the macros, data structure and prototypes of the touch controller MTCH6301 driver interface.

File Name

drv_mtch6301.c

drv_mtch6301_config_template.h

MTCH6301 Touch Driver configuration template.

Macros

	Name	Description
	DRV_MTCH6301_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_MTCH6301_CALIBRATION_INSET	Defines the calibration inset.
	DRV_MTCH6301_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_MTCH6301_INDEX	MTCH6301 static index selection.
	DRV_MTCH6301_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_MTCH6301_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_MTCH6301_SAMPLE_POINTS	Define the sample points.
	DRV_MTCH6301_TOUCH_DIAMETER	Defines the touch diameter.

Description

MTCH6301 Touch Driver Configuration Template

This header file contains the build-time configuration selections for the MTCH6301 Touch Driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_mtch6301_config_template.h

Company

Microchip Technology Inc.

MTCH6303 Touch Driver Library

This topic describes the MTCH6303 Touch Driver Library.

Introduction

This library provides an interface to manage the MTCH6303 Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony MTCH6303 Touch Driver provides a high-level interface to the MTCH6303 touch controller device. This driver provides application routines to read the touch input data from the touch screen. The MTCH6303 device can notify the availability of touch input data through external interrupt. The MTCH6303 driver allows the application to map a controller pin as an external interrupt pin.

Currently, the MTCH6303 Touch Driver only supports non-gestural single-finger touch screen input.

Using the Library

This topic describes the basic architecture of the MTCH6303 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_mtch6303_static.h`

The interface to the MTCH6303 Touch Driver Library is defined in the `drv_mtch6303_static.h` header file. This file is generated by the MPLAB Harmony Configurator (MHC) during application code generation. It is included in `system_definitions.h` by MHC during application code generation. Any configuration macros required for MTCH6303 Driver are included in `system_config.h` by MHC during code generation. Any C language source (`.c`) file that uses the MTCH6303 Touch Driver Library should include `system_config.h` and `system_definitions.h`, respectively.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MTCH6303 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

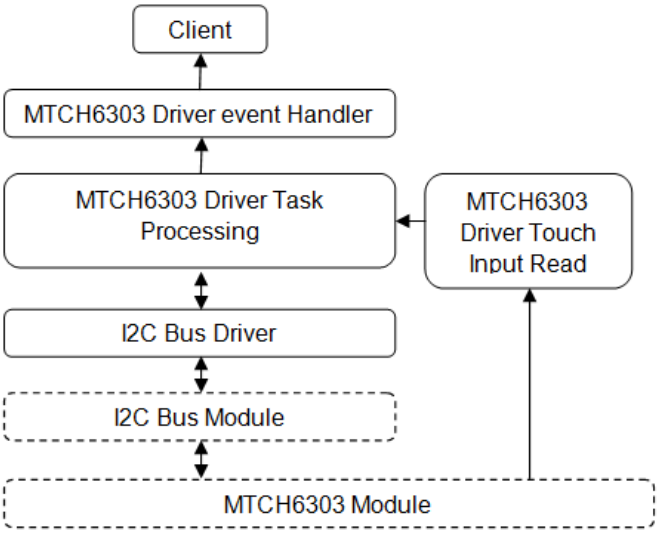
The MTCH6303 Touch Driver has routines to perform the following operations:

- MTCH6303 register read and write
- MTCh6303 message read and write
- MTCH6303 touch input read
- Mapping of the touch input to screen resolution

The driver Initialization routine allows the application to initialize the driver. The driver must be initialized before it can be used by the application. Once the driver is initialized, the driver Open function allows retrieval of the client handle. If the client handle is valid, an event handler routine needs to be registered by the application. The MTCH6303 Touch Driver triggers an interrupt once touch input is available to be read from the MTCH6303 registers. A touch input Read function is called from the interrupt handler to initiate the touch input read task. An Event Handler function is called once the touch input read task is completed. A valid touch input will be available only after the event handler routine is triggered. The touch input must be read inside of the event handler function.

The touch input data is a raw value and needs to be mapped to the target screen resolution. At zero degree orientation, touch input is mapped on the x axis from zero at the left and the maximum value at the right. At zero degree orientation, touch input is mapped on the y axis from zero at the top and the maximum value at the bottom.

MTCH6303 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MTCH6303 Touch Driver.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks, and status functions.
Client Setup Functions	Provides open, close, status, and other setup function.
Read and Write Functions	Provides functions to read and write to the MTCH6303 registers, messages, and touch data.
Miscellaneous Functions	Provides miscellaneous functions.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, task, and status functions
- Client setup functions, which provide client interfaces such as open, close and event handler registration
- Read and write functions, initiate the touch input or register or message read and write tasks
- Miscellaneous functions such as touch input map function are provided to process the raw touch input data

Configuring the Library

The configuration of the MTCH6303 Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the MTCH6303 Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the MTCH6303 Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the MTCH6303 Touch Driver Library.

Description

This section list the files that are available in the `/src` folder of the MTCH6303 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/mtch6303`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_mtch6303_static.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/drv_mtch6303_static.c`	Basic MTCH6303 Touch Driver implementation file.
`/src/drv_mtch6303_buffer_queue_i2c_static.c`	MTCH6303 I2C buffer queue implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/drv_mtch6303_buffer_queue_touch_static.c	MTCH6303 message buffer queue implementation file.

Module Dependencies

The MTCH6303 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_MTCH6303_Deinitialize	Deinitializes the instance of the MTCH6303 driver module.
	DRV_MTCH6303_Initialize	Initializes the MTCH6303 static single instance.
	DRV_MTCH6303_Status	Gets the current status of the MTCH6303 driver module.
	DRV_MTCH6303_Tasks	Maintains the driver's register read/write state machine and implements its ISR.

b) Client Setup Functions

	Name	Description
	DRV_MTCH6303_Close	Closes an opened-instance of the MTCH6303 driver.
	DRV_MTCH6303_ErrorGet	This function returns the error associated with the last client request.
	DRV_MTCH6303_Open	Opens the MTCH6303 driver instance and returns a handle to it.

c) Read and Write Functions

	Name	Description
	DRV_MTCH6303_AddRegisterRead	Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.
	DRV_MTCH6303_AddRegisterWrite	Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageCommandWrite	Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageReportRead	Schedules a non-blocking report message read request to read the report message from MTCH6303 device.
	DRV_MTCH6303_TOUCH_AddTouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_TOUCH_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.
	DRV_MTCH6303_TOUCH_Tasks	Maintains the driver's message state machine and implements its ISR.
	DRV_MTCH6303_TouchInputMap	Maps the raw touch input to display resolution.
	DRV_MTCH6303_TouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

d) Data Types and Constants

	Name	Description
	DRV_MTCH6303_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_NUM_INPUTS	Definition of number of touch input packets can be identified by MTCH6303.
	DRV_MTCH6303_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.
	DRV_MTCH6303_BUFFER_EVENT_HANDLER	Points to a callback after completion of an register read -write or message stream read - write.
	DRV_MTCH6303_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.

	DRV_MTCH6303_CLIENT_STATUS	Defines the client-specific status of the MTCH6303 driver.
	DRV_MTCH6303_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT	Lists the different conditions that happens during a touch message buffer transfer.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER	Points to a callback after completion of an message report read or message command write.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE	Handle identifying a read or write touch message buffer passed to the driver.
	DRV_MTCH6303_TOUCH_DATA	Defines MTCH6303 I2C Touch Data
	DRV_MTCH6303_TOUCH_INPUT	Defines MTCH6303 Touch Input Packet
	DRV_MTCH6303_TOUCH_MESSAGE	Defines MTCH6303 Touch Message.
	DRV_MTCH6303_TOUCH_MESSAGE_HEADER	Defines Touch Message Header.
	DRV_MTCH6303_TOUCH_NIBBLE_0	Defines the I2C Nibble 0 of MTCH6303 Touch input packet.
	DRV_MTCH6303_TOUCH_STATUS	Defines the I2C touch status register bits
	DRV_TOUCH_MTCH6303_MSG_ID	List of report or command message identification.
	DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP	List of MTCH6303 I2C Accessible Register Identification.

Description

This section describes the API functions of the MTCH6303 Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_MTCH6303_Deinitialize Function

Deinitializes the instance of the MTCH6303 driver module.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_Deinitialize();
```

Returns

None.

Description

Deinitializes the instance of the MTCH6303 driver module, disabling its operation. Invalidates all the internal data.

Remarks

once the initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. this routine will NEVER block waiting for hardware.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function.

Example

```
SYS_STATUS    status;

DRV_MTCH6303_Deinitialize();

status = DRV_MTCH6303_Status();
if(SYS_MODULE_DEINITIALIZED != status)
{
    //check again later if you need to know
    //when the driver is deinitialized
}
```

Function

void DRV_MTCH6303_Deinitialize(void)

DRV_MTCH6303_Initialize Function

Initializes the MTCH6303 static single instance.

File

drv_mtch6303.h

C

```
SYS_MODULE_OBJ DRV_MTCH6303_Initialize();
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MTCH6303 static driver instance. It makes the instance ready for a client to open and use it. The instance parameters are initialized by values set by MPLAB Harmony Configurator.

Preconditions

None.

Example

```
// The following code snippet shows an example MTCH6303 driver initialization.

SYS_MODULE_OBJ    objectHandle;

objectHandle = DRV_MTCH6303_Initialize();
if( SYS_MODULE_OBJ_INVALID == objectHandle )
{
   // Handle error
}
Remarks:   This routine must be called before any other MTCH6303 routine is called.

   This routine should only be called once during system initialization unless
   DRV_MTCH6303_Deinitialize is called to deinitialize the driver instance.
   This routine will NEVER block for hardware access.
```

Function

SYS_MODULE_OBJ DRV_MTCH6303_Initialize (void)

DRV_MTCH6303_Status Function

Gets the current status of the MTCH6303 driver module.

File

drv_mtch6303.h

C

```
SYS_STATUS DRV_MTCH6303_Status();
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another.

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized.

Description

This routine provides the current status of the MTCH6303 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function.

Example

```
SYS_STATUS mtch6303Status;

mtch6303Status = DRV_MTCH6303_Status();
if(SYS_STATUS_READY == mtch6303Status)
{
    // This means the driver can be opened using the
    // DRV_MTCH6303_Open() function.
}
```

Function

SYS_STATUS DRV_MTCH6303_Status(void)

DRV_MTCH6303_Tasks Function

Maintains the driver's register read/write state machine and implements its ISR.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_Tasks();
```

Returns

None.

Description

This routine is used to maintain the driver's register read/write state machine and implement its ISR for interrupt-driven implementations. In interrupt mode, this function is called in I2C Driver event Handler routine. The I2C Driver event Handler routine is registered by MTCH6303 event Handler register routine.

Remarks

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function. It also needs registration of the MTCH6303 Driver event handler routine.

Function

void DRV_MTCH6303_Tasks(void)

b) Client Setup Functions

DRV_MTCH6303_Close Function

Closes an opened-instance of the MTCH6303 driver.

File

drv_mtch6303.h

C

```
DRV_MTCH6303_CLIENT_STATUS DRV_MTCH6303_Close();
```

Returns

DRV_MTCH6303_CLIENT_STATUS_ERROR - if driver fails to remove buffer objects from queue.

DRV_MTCHC6303_CLIENT_STATUS_CLOSED - client is successfully closed

Description

This routine closes an opened-instance of the MTCH6303 driver. Any buffers in the driver queue that were submitted by this client will be removed. DRV_MTCH6303_Open must be called to before using the driver again.

Remarks

The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called. DRV_MTCH6303_Open must have been called.

Example

```
DRV_MTH6303_CLIENT_STATUS mtch6303Status;

mtch6303Status = DRV_MTCH6303_Close()
if( DRV_MTCH6303_CLIENT_STATUS_ERROR == mtch6303Status )
{
    //retry closing the driver client
}
```

Function

DRV_MTCH6303_CLIENT_STATUS DRV_MTCH6303_Close (void)

DRV_MTCH6303_ErrorGet Function

This function returns the error associated with the last client request.

File

drv_mtch6303.h

C

```
DRV_MTCH6303_ERROR DRV_MTCH6303_ErrorGet();
```

Returns

DRV_MTCH6303_ERROR_NONE - no error

Description

This function returns the error associated with the last client request.

Remarks

This routine always return DRV_MTCH6303_ERROR_NONE the client error is currently not updated by any of the MTCH6303 operations API's.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called. DRV_MTCH6303_Open must have been called to open a device client.

Function

DRV_MTCH6303_ERROR DRV_MTCH6303_ErrorGet (void)

DRV_MTCH6303_Open Function

Opens the MTCH6303 driver instance and returns a handle to it.

File

drv_mtch6303.h

C

```
DRV_HANDLE DRV_MTCH6303_Open();
```

Returns

If successful, the routine returns a valid open-instance handle. If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the driver is not ready to be opened, typically when the initialize routine has not completed execution.
- if the bus driver fails to open
- if the client is trying to open the driver but driver has been opened exclusively by another client.

Description

This routine opens the specified MTCH6303 driver instance and provides a handle.

Remarks

The handle returned is valid until the DRV_MTCH6303_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_MTCH6303_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_MTCH6303_Open( );
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Function

DRV_HANDLE DRV_MTCH6303_Open { void }

c) Read and Write Functions

DRV_MTCH6303_AddRegisterRead Function

Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_AddRegisterRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, uint8_t source, size_t nBytes,
uint8_t * destination);
```

Returns

None.

Description

This function schedules a non-blocking register read request to read I2C accessible MTCH6303 registers. The function returns with a valid buffer handle in the bufferHandle argument if the register read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The register data is collected into destination and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the register data must be read from destination.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t registerData[NUM_REGISTERS];
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver
```

```
DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);

DRV_MTCH6303_AddRegisterRead( &bufferHandle,
                              DRV_MTCH6303_REG_TOUCH_STATUS,
                              NUM_REGISTERS,
                              &registerData );

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
Remarks:   None.
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
source	Register index.
nBytes	Number of registers to be read, starting from source.
destination	buffer collecting register data.

Function

void DRV_MTCH6303_AddRegisterRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,

uint8_t source,

size_t nBytes,

uint8_t * destination)

DRV_MTCH6303_AddRegisterWrite Function

Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_AddRegisterWrite(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, uint8_t destination, size_t
nBytes, uint8_t * source);
```

Returns

None.

Description

This function schedules a non-blocking register write request to write I2C accessible MTCH6303 registers. The function returns with a valid buffer handle in the bufferHandle argument if the register write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. A event handler is called on buffer event complete where the application data is written to the I2C accessible MTCH6303 Register.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t registerData[NUM_REGISTERS];
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);

DRV_MTCH6303_AddRegisterWrite( &bufferHandle,
                               DRV_MTCH6303_REG_TOUCH_STATUS,
                               NUM_REGISTERS,
                               &registerData );

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;
```

```
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
bufferHandle	Pointer to an argument that will contain the return buffer handle.
destination	Index to the start of destination register list.
nBytes	number of registers.
source	pointer to the data to be written to the register.

Function

void DRV_MTCH6303_AddRegisterWrite(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,

uint8_t destination,

size_t nBytes,

uint8_t * source)

DRV_MTCH6303_TOUCH_AddMessageCommandWrite Function

Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddMessageCommandWrite(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageCmd, size_t messageSize);
```

Returns

None.

Description

This function schedules a non-blocking command message write request to write command message to MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the register command message write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application message buffer is owned by the driver and should not be modified. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the message write queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. A event handler is called on buffer event complete where the application command message is written to MTCH6303.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageCommand;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver
```

```
DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddMessageCommandWrite( &bufferHandle,
                                           &messageCommand,
                                           MY_MESSAGE_SIZE );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Pointer to an argument that will contain the return buffer handle.
messageCmd	command message to write to MTCH6303.
messageSize	command message size. It includes message header and payload size.

Function

void DRV_MTCH6303_TOUCH_AddMessageCommandWrite
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageCmd,
size_t messageSize)

DRV_MTCH6303_TOUCH_AddMessageReportRead Function

Schedules a non-blocking report message read request to read the report message from MTCH6303 device.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddMessageReportRead(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageRep, size_t messageSize);
```

Returns

None.

Description

This function schedules a non-blocking report message read request to read the report message from MTCH6303 device. The function returns with a valid buffer handle in the bufferHandle argument if the register read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The register data is collected into destination and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the register data must be read from destination.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageReport;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddMessageReportRead( &bufferHandle,
                                         &messageReport,
                                         MY_MESSAGE_SIZE );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
```

```
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
messageRep	report message buffer.
messageSize	report message size. It includes message header and payload size.

Function

void DRV_MTCH6303_TOUCH_AddMessageReportRead
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageRep,
size_t messageSize)

DRV_MTCH6303_TOUCH_AddTouchInputRead Function

Schedules a non-blocking read buffer request to read touch input from MTCH6303.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddTouchInputRead(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData);
```

Returns

None.

Description

This function schedules a non-blocking read buffer request to read touch input from MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The touch data is collected into touchData and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the touch data must be read from touchData.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_DATA touchData;
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddTouchInputRead( &bufferHandle, &touchData );
```

```
if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
touchData	Buffer collecting touch data.

Function

void DRV_MTCH6303_TOUCH_AddTouchInputRead
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData)

DRV_MTCH6303_TOUCH_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_BufferEventHandlerSet(const DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a message event handling function for the driver to call back when queued message transfers have finished. When a client calls either the DRV_MTCH6303_TOUCH_AddTouchInputRead, DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite function, it is provided with a handle identifying the message that was added to the driver's message queue. The driver will pass this handle back to the client by calling "eventHandler" function when the message transfer has completed.

The event handler should be set before the client performs any "message add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageReport;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// myMTCH6303Handle is the handle returned
// by the DRV_MTCH6303_Open function.

// Client registers an event handler with driver. This is done once

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                (uintptr_t)&myAppObj );

DRV_MTCH6303_TOUCH_AddMessageReportRead( &bufferHandle, &messageReport );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_MTCH6303_TOUCH_BufferEventHandlerSet

(

const DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context
)

DRV_MTCH6303_TOUCH_Tasks Function

Maintains the driver's message state machine and implements its ISR.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_Tasks();
```

Returns

None.

Description

This routine is used to maintain the driver's message state machine and implement its ISR for interrupt-driven implementations. In interrupt mode, this function is called in I2C Driver event Handler routine. The I2C Driver event Handler routine is registered by MTCH6303 Touch event Handler register routine.

Remarks

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function. It also needs registration of the MTCH6303 Driver Touch event handler routine.

Function

void DRV_MTCH6303_TOUCH_Tasks(void)

DRV_MTCH6303_TouchInputMap Function

Maps the raw touch input to display resolution.

File

drv_mtch6303.h

C

```
inline uint16_t DRV_MTCH6303_TouchInputMap(uint16_t touchValue, uint16_t dispResolution);
```

Returns

This function returns the raw touch input mapped to display resolution in form of number of pixels.

Description

This function maps the raw touch input to display resolution. Raw touch input touchValue is obtained from the individual x or y value of DRV_MTCH6303_TOUCH_DATA. Raw touch value varies from 0 to 0x7FFF. The displayResolution is either horizontal or vertical resolution of the display in pixels. The function returns the raw touch input mapped to display resolution in form of number of pixels.

Remarks

None.

Preconditions

None.

Example

```
// Display with resolution 800 x 480
#define DISP_HOR_RESOUTION  800
#define DISP_VER_RESOLUTION 480

DRV_MTCH6303_TOUCH_DATA touchData;
uint16_t rawTouchX;
```

```
uint16_t rawTouchY;
uint16_t touchX;
uint16_t touchY;

// map 0th touch packet to display resolution
rawTouchX = touchData.touch[0].x;
rawTouchY = touchData.touch[0].y;

// map raw touch input in x direction to display horizontal resolution
touchX = DRV_MTCH6303_TouchInputMap( rawTouchX, DISP_HOR_RESOLUTION );

// map raw touch input in y direction to display vertical resolution
touchY = DRV_MTCH6303_TouchInputMap( rawTouchY, DISP_VER_RESOLUTION );

// use touchX and touchY as input to graphics objects.
```

Parameters

Parameters	Description
touchValue	raw touch input either in x or y direction (0 - 0x7FFF).
dispResolution	display resolution specifying either width or height of the display in pixels.

Function

uint16_t DRV_MTCH6303_TouchInputMap(uint16_t touchValue, uint16_t dispResolution)

DRV_MTCH6303_TouchInputRead Function

Schedules a non-blocking read buffer request to read touch input from MTCH6303.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TouchInputRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, DRV_MTCH6303_TOUCH_DATA *
touchData);
```

Returns

None.

Description

This function schedules a non-blocking read buffer request to read touch input from MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The touch data is collected into touchData and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the touch data must be read from touchData.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_DATA touchData;
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;
```

```
// Client registers an event handler with driver

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);

DRV_MTCH6303_TouchInputRead( &bufferHandle, &touchData );

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
touchData	Buffer collecting touch data.

Function

void DRV_MTCH6303_TouchInputRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData)

DRV_MTCH6303_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_BufferEventHandlerSet(const DRV_MTCH6303_BUFFER_EVENT_HANDLER eventHandler, const
uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_MTCH6303_TouchInputRead, DRV_MTCH6303_AddRegisterRead or DRV_MTCH6303_AddRegisterWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by

calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myMTCH6303Handle is the handle returned
// by the DRV_MTCH6303_Open function.

// Client registers an event handler with driver. This is done once

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandle,
                                    (uintptr_t)&myAppObj );

DRV_MTCH6303_AddRegisterRead( &bufferHandle
                              DRV_MTCH6303_REG_TOUCH_STATUS,
                              MY_BUFFER_SIZE,
                              &mybuffer);

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandle( DRV_MTCH6303_BUFFER_EVENT event,
                                    DRV_MTCH6303_BUFFER_HANDLE handle,
                                    uintptr_t context)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
eventHandler	Pointer to the event handler function.

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_MTCH6303_BufferEventHandlerSet
(
const DRV_MTCH6303_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context
)

d) Data Types and Constants

DRV_MTCH6303_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_BUFFER_HANDLE_INVALID
```

Description

MTCH6303 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_MTCH6303_AddRegisterRead, DRV_MTCH6303_AddRegisterWrite or DRV_MTCH6303_TouchInputRead functions if the request was not successful.

Remarks

None

DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID
```

Description

MTCH6303 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_MTCH6303_TOUCH_AddMessageReportRead, DRV_MTCH6303_TOUCH_AddMessageCommandWrite or DRV_MTCH6303_TOUCH_AddTouchInputRead functions if the request was not successful.

Remarks

None

DRV_MTCH6303_TOUCH_NUM_INPUTS Macro

Definition of number of touch input packets can be identified by MTCH6303.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_TOUCH_NUM_INPUTS 0xA
```

Description

MTCH6303 Number of touch input packets

MTCH6303 supports multi-touch and can identify upto 10 different touch input packets.

Remarks

None.

DRV_MTCH6303_BUFFER_EVENT Enumeration

Lists the different conditions that happens during a buffer transfer.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_BUFFER_EVENT_COMPLETE,
  DRV_MTCH6303_BUFFER_EVENT_ERROR,
  DRV_MTCH6303_BUFFER_EVENT_ABORT
} DRV_MTCH6303_BUFFER_EVENT;
```

Members

Members	Description
DRV_MTCH6303_BUFFER_EVENT_COMPLETE	Event buffer transfer complete
DRV_MTCH6303_BUFFER_EVENT_ERROR	Event buffer transfer error
DRV_MTCH6303_BUFFER_EVENT_ABORT	Event buffer transfer abort

Description

MTCH6303 Buffer Events

This enumeration identifies the different conditions that can happen during a buffer transaction. Callbacks can be made with the appropriate buffer condition passed as a parameter to execute the desired action.

The values act like flags and multiple flags can be set.

Remarks

None.

DRV_MTCH6303_BUFFER_EVENT_HANDLER Type

Points to a callback after completion of an register read -write or message stream read - write.

File

drv_mtch6303.h

C

```
typedef void (* DRV_MTCH6303_BUFFER_EVENT_HANDLER)(DRV_MTCH6303_BUFFER_EVENT event,
DRV_MTCH6303_BUFFER_HANDLE bufferHandle, uintptr_t context);
```

Description

MTCH6303 Buffer Event Callback

This type identifies the MTCH6303 Buffer Event. It allows the client driver to register a callback using DRV_MTCH6303_BUFFER_EVENT_HANDLER. By using this mechanism, the driver client will be notified at the completion of the corresponding transfer.

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_MTCH6303_BUFFER_EVENT	Status of MTCH6303 transfer

bufferHandle	Handle that identifies the particular Buffer Object
context	pointer to the object to be processed.

Function

void (*DRV_MTCH6303_BUFFER_EVENT_HANDLER) (DRV_MTCH6303_BUFFER_EVENT event,
DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
uintptr_t context)

DRV_MTCH6303_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_mtch6303.h

C

```
typedef uintptr_t DRV_MTCH6303_BUFFER_HANDLE;
```

Description

MTCH6303 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_MTCH6303_AddRegisterRead, DRV_MTCH6303_AddRegisterWrite or DRV_MTCH6303_TouchInputRead functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from these functions is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_MTCH6303_CLIENT_STATUS Enumeration

Defines the client-specific status of the MTCH6303 driver.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR,
  DRV_MTCH6303_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_MTCH6303_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_MTCH6303_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY
} DRV_MTCH6303_CLIENT_STATUS;
```

Members

Members	Description
DRV_MTCH6303_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	An error has occurred.
DRV_MTCH6303_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_MTCH6303_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	The driver is currently busy and cannot start additional operations.
DRV_MTCH6303_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY	The module is running and ready for additional operations

Description

MTCH6303 Client-Specific Driver Status

This enumeration defines the client-specific status codes of the MTCH6303 driver.

Remarks

Returned by the DRV_MTCH6303_ClientStatus function.

DRV_MTCH6303_ERROR Enumeration

Defines the possible errors that can occur during driver operation.

File

drv_mtch6303.h

C

```
typedef enum {
} DRV_MTCH6303_ERROR;
```

Description

MTCH6303 Driver Errors.

This data type defines the possible errors that can occur when occur during MTCH6303 driver operation. These values are returned by DRV_MTCH6303_ErrorGet function.

Remarks

None

DRV_MTCH6303_TOUCH_BUFFER_EVENT Enumeration

Lists the different conditions that happens during a touch message buffer transfer.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE,
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR,
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_ABORT
} DRV_MTCH6303_TOUCH_BUFFER_EVENT;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE	Event touch message buffer transfer complete
DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR	Event touch message buffer transfer error
DRV_MTCH6303_TOUCH_BUFFER_EVENT_ABORT	Event touch message buffer transfer abort

Description

MTCH6303 Touch Message Buffer Events

This enumeration identifies the different conditions that can happen during a touch message buffer transaction. Callbacks can be made with the appropriate touch message buffer condition passed as a parameter to execute the desired action.

The values act like flags and multiple flags can be set.

Remarks

None.

DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER Type

Points to a callback after completion of an message report read or message command write.

File

drv_mtch6303.h

C

```
typedef void (* DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER)(DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle, uintptr_t context);
```

Description

MTCH6303 Touch Buffer Event Callback

This type identifies the MTCH6303 Touch Buffer Event. It allows the client driver to register a callback using DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER. By using this mechanism, the driver client will be notified at the completion of the corresponding transfer.

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_MTCH6303_TOUCH_BUFFER_EVENT	Status of MTCH6303 touch message transfer
bufferHandle	Handle that identifies the particular Buffer Object
context	pointer to the object to be processed.

Function

void (*DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER) (DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
uintptr_t context)

DRV_MTCH6303_TOUCH_BUFFER_HANDLE Type

Handle identifying a read or write touch message buffer passed to the driver.

File

drv_mtch6303.h

C

```
typedef uintptr_t DRV_MTCH6303_TOUCH_BUFFER_HANDLE;
```

Description

MTCH6303 Driver Touch Message Queue Buffer Handle

A touch message buffer handle value is returned by a call to the DRV_MTCH6303_TOUCH_AddMessageReportRead, DRV_MTCH6303_TOUCH_AddMessageCommandWrite or DRV_MTCH6303_TOUCH_AddTouchInputRead. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from these functions is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_MTCH6303_TOUCH_DATA Structure

Defines MTCH6303 I2C Touch Data

File

drv_mtch6303.h

C

```
typedef struct {
  uint8_t i2cReadAddr;
  DRV_MTCH6303_TOUCH_STATUS status;
  DRV_MTCH6303_TOUCH_INPUT touch[ DRV_MTCH6303_TOUCH_NUM_INPUTS ];
} DRV_MTCH6303_TOUCH_DATA;
```

Members

Members	Description
uint8_t i2cReadAddr;	Dummy I2C Read Address required for bitbang driver
DRV_MTCH6303_TOUCH_STATUS status;	MTCH6303 Touch Status
DRV_MTCH6303_TOUCH_INPUT touch[DRV_MTCH6303_TOUCH_NUM_INPUTS];	MTCH6303 Touch Input array of size DRV_MTCH6303_TOUCH_NUM_INPUTS

Description

MTCH6303 I2C Touch Data

This structure defines MTCH6303 I2C Touch Data. The structure DRV_MTCH6303_TOUCH_DATA is passed to API's DRV_MTCH6303_AddRegisterRead or DRV_MTCH6303_TOUCH_AddTouchInputRead. The API's will update the structure with touch input.

Remarks

It is packed to form structure of size 62 bytes. The structure member i2cReadAddr is only applicable if the I2C driver is of type bitbang. Otherwise the variable required to be commented out.

DRV_MTCH6303_TOUCH_INPUT Structure

Defines MTCH6303 Touch Input Packet

File

drv_mtch6303.h

C

```
typedef struct {
  DRV_MTCH6303_TOUCH_NIBBLE_0 nibble_0;
  uint8_t touchId;
  uint16_t x;
  uint16_t y;
} DRV_MTCH6303_TOUCH_INPUT;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_NIBBLE_0 nibble_0;	MTCH6303 I2C Touch Input Packet Nibble 0
uint8_t touchId;	MTCH6303 I2C Touch Input Packet ID (0 - 16)
uint16_t x;	MTCH6303 I2C Touch Input Packet position x (0 - 0x7FFF)
uint16_t y;	MTCH6303 I2C Touch Input Packet position y (0 - 0x7FFF)

Description

MTCH6303 Touch Input Packet.

This structure defines the MTCH6303 Touch Input Packet.

Remarks

It is part of DRV_MTCH6303_TOUCH_DATA structure. It is packed to form structure of size 6 bytes.

DRV_MTCH6303_TOUCH_MESSAGE Structure

Defines MTCH6303 Touch Message.

File

drv_mtch6303.h

C

```
typedef struct {
  DRV_MTCH6303_TOUCH_MESSAGE_HEADER header;
  uint8_t payload[0x3E];
} DRV_MTCH6303_TOUCH_MESSAGE;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_MESSAGE_HEADER header;	MTCH6303 Touch Message Header
uint8_t payload[0x3E];	MTCH6303 Touch Message payload. First byte of payload is of type DRV_TOUCH_MTCH6303_MSG_ID in case of first fragment of message. Otherwise the first byte acts as a normal payload.

Description

MTCH6303 Touch Message

This structure defines MTCH6303 Touch Message. The variable pointer of type DRV_MTCH6303_TOUCH_MESSAGE is passed to the API's

DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite.

Remarks

It is packed to form structure of size 63 bytes.

DRV_MTCH6303_TOUCH_MESSAGE_HEADER Structure

Defines Touch Message Header.

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t msgFragSize : 6;
  uint32_t continued : 1;
  uint32_t moreMessages : 1;
} DRV_MTCH6303_TOUCH_MESSAGE_HEADER;
```

Members

Members	Description
uint32_t msgFragSize : 6;	MTCH6303 Message Fragment Size. If Message Fragment size is 0x3F the Fragment is incomplete and uses up ALL of the parent transport layer packet.
uint32_t continued : 1;	MTCH6303 Message continued from last fragment if set to 1.
uint32_t moreMessages : 1;	MTCH6303 more messages to follow in this block if set to 1.

Description

MTCH6303 Touch Message Header

This structure defines Touch Message Header.

Remarks

It is part of structure DRV_MTCH6303_TOUCH_MESSAGE. It is packed to form structure of size 1 byte.

DRV_MTCH6303_TOUCH_NIBBLE_0 Structure

Defines the I2C Nibble 0 of MTCH6303 Touch input packet.

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t touchState : 1;
  uint32_t inRange : 1;
  uint32_t reserved : 6;
} DRV_MTCH6303_TOUCH_NIBBLE_0;
```

Members

Members	Description
uint32_t touchState : 1;	Touch packet available
uint32_t inRange : 1;	Touch packet in range
uint32_t reserved : 6;	Reserved bits

Description

MTCH6303 I2C Touch Input Packet Nibble 0

This structure defines the I2C Nibble 0 of MTCH6303 Touch input packet.

Remarks

It is part of DRV_MTCH6303_TOUCH_INPUT structure. It is packed to form structure of size 1 byte.

DRV_MTCH6303_TOUCH_STATUS Structure

Defines the I2C touch status register bits

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t nTouch : 4;
  uint32_t streamReady : 1;
  uint32_t gestureReady : 1;
  uint32_t gestICData : 1;
  uint32_t reserved : 1;
} DRV_MTCH6303_TOUCH_STATUS;
```

Members

Members	Description
uint32_t nTouch : 4;	Number of available touch packets
uint32_t streamReady : 1;	stream data ready
uint32_t gestureReady : 1;	gesture data ready
uint32_t gestICData : 1;	GestIC data ready
uint32_t reserved : 1;	reserved bit

Description

MTCH6303 I2C touch status

This structure defines the I2C touch status register bits.

Remarks

It is part of DRV_MTCH6303_TOUCH_DATA structure. It is packed to form structure of size 1 byte.

DRV_TOUCH_MTCH6303_MSG_ID Enumeration

List of report or command message identification.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_TOUCH_MTCH6303_MSG_CMD_QUERY_VERSION
} DRV_TOUCH_MTCH6303_MSG_ID;
```

Members

Members	Description
DRV_TOUCH_MTCH6303_MSG_CMD_QUERY_VERSION	Message sends firmware version query command. Bytes 124:127 = Rev[2].Minor.Major

Description

MTCH6303 Touch message Identification.

This enumeration identifies the different report or command messages supported by MTCH6303. This identifier identifies the type of the message. The identifier is passed in the message DRV_MTCH6303_TOUCH_MESSAGE as first byte of the payload. It is applicable only for first fragment of message. If message involves multiple fragments, the payload of message fragments other than first fragment should start with normal payload byte. The touch message is read or send to MTCH6303 by using DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite.

Remarks

To be passed as first byte of message payload. Applicable only for first fragment of message.

DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP Enumeration

List of MTCH6303 I2C Accessible Register Identification.

File

drv_mtch6303.h

C

```
typedef enum {
} DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP;
```

Description

MTCH6303 I2C Accessible Register Identification.

This enumeration identifies the different I2C accessible MTCH6303 Registers. The identifier is passed as source to the register read routine or as destination to the register write routine. The MTCH6303 driver routine to read the I2C accessible MTCH6303 registers is DRV_MTCH6303_AddRegisterRead. The MTCH6303 driver routine to write the I2C accessible MTCH6303 registers is DRV_MTCH6303_AddRegisterWrite.

Remarks

To read or write multiple registers, identifier of only first register is sufficient as source or destination respectively.

Files

Files

Name	Description
drv_mtch6303.h	MTCH6303 driver interface declarations for the static single instance driver.

Description

This section lists the source and header files used by the MTCH6303 Touch Driver Library.

drv_mtch6303.h

MTCH6303 driver interface declarations for the static single instance driver.

Enumerations

	Name	Description
	DRV_MTCH6303_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.
	DRV_MTCH6303_CLIENT_STATUS	Defines the client-specific status of the MTCH6303 driver.
	DRV_MTCH6303_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT	Lists the different conditions that happens during a touch message buffer transfer.
	DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP	List of MTCH6303 I2C Accessible Register Identification.
	DRV_TOUCH_MTCH6303_MSG_ID	List of report or command message identification.

Functions

	Name	Description
	DRV_MTCH6303_AddRegisterRead	Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.
	DRV_MTCH6303_AddRegisterWrite	Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.
	DRV_MTCH6303_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_MTCH6303_Close	Closes an opened-instance of the MTCH6303 driver.
	DRV_MTCH6303_Deinitialize	Deinitializes the instance of the MTCH6303 driver module.
	DRV_MTCH6303_ErrorGet	This function returns the error associated with the last client request.
	DRV_MTCH6303_Initialize	Initializes the MTCH6303 static single instance.
	DRV_MTCH6303_Open	Opens the MTCH6303 driver instance and returns a handle to it.
	DRV_MTCH6303_Status	Gets the current status of the MTCH6303 driver module.

	DRV_MTCH6303_Tasks	Maintains the driver's register read/write state machine and implements its ISR.
	DRV_MTCH6303_TOUCH_AddMessageCommandWrite	Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageReportRead	Schedules a non-blocking report message read request to read the report message from MTCH6303 device.
	DRV_MTCH6303_TOUCH_AddTouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_TOUCH_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.
	DRV_MTCH6303_TOUCH_Tasks	Maintains the driver's message state machine and implements its ISR.
	DRV_MTCH6303_TouchInputMap	Maps the raw touch input to display resolution.
	DRV_MTCH6303_TouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.

Macros

	Name	Description
	DRV_MTCH6303_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_NUM_INPUTS	Definition of number of touch input packets can be identified by MTCH6303.

Structures

	Name	Description
	DRV_MTCH6303_TOUCH_DATA	Defines MTCH6303 I2C Touch Data
	DRV_MTCH6303_TOUCH_INPUT	Defines MTCH6303 Touch Input Packet
	DRV_MTCH6303_TOUCH_MESSAGE	Defines MTCH6303 Touch Message.
	DRV_MTCH6303_TOUCH_MESSAGE_HEADER	Defines Touch Message Header.
	DRV_MTCH6303_TOUCH_NIBBLE_0	Defines the I2C Nibble 0 of MTCH6303 Touch input packet.
	DRV_MTCH6303_TOUCH_STATUS	Defines the I2C touch status register bits

Types

	Name	Description
	DRV_MTCH6303_BUFFER_EVENT_HANDLER	Points to a callback after completion of an register read -write or message stream read - write.
	DRV_MTCH6303_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER	Points to a callback after completion of an message report read or message command write.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE	Handle identifying a read or write touch message buffer passed to the driver.

Description

MTCH6303 Driver Interface Declarations for Static Single Instance Driver

The MTCH6303 device driver provides a simple interface to manage the MTCH6303 module. This file defines the interface Declarations for the MTCH6303 driver.

Remarks

Static single instance driver interface eliminates the need for an object ID or object handle. Static single-open interfaces also eliminate the need for the open handle.

File Name

drv_mtch6303_static.h

Company

Microchip Technology Inc.

mXT336T Touch Driver Library

This topic describes the mXT336T Touch Driver Library.

Introduction

This library provides an interface to manage the mXT336T Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony mXT336T Touch Driver provides a high-level interface to the mXT336T touch controller device. This driver provides application routines to read the touch input data from the touch screen. The mXT336T device can notify the availability of touch input data through external interrupt. The mXT336T driver allows the application to map a controller pin as an external interrupt pin.

Currently, the mXT336T Touch Driver only supports non-gestural single-fingered touch input.

Using the Library

This topic describes the basic architecture of the mXT336T Touch Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_mxt336t.h`, `drv_mxt.h`

The interface to the mXT336T Touch Driver library is defined in the `drv_mxt336t.h` and `drv_mxt.h` header files. Any C language source (`.c`) file that uses the mXT336T Touch Driver library should include this header.

The mXT336T Touch Driver is based on the Object Protocol for the Atmel® maXTouch® mXT336T Touchscreen Controller.

The functioning of the driver is divided into two file sets:

- `drv_mxt.h` has the system touch interface (API's, Initialization and tasks)
- `drv_mxt336t.h` has the device specific interface for getting the device ready for communication and receiving commands.

The device specific interface is based on the Object Protocol previously mentioned.

The aria_quickstart demonstration interfaces with the mXT336T Touch Driver Library. Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the mXT336T Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

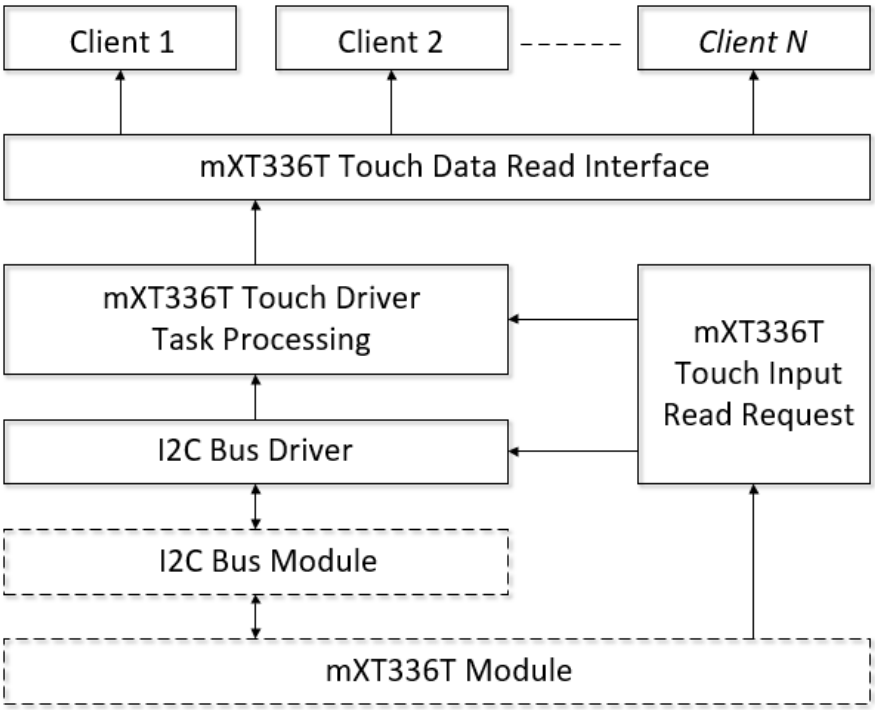
Description

The mXT336T Touch Driver has routines to perform the following operations:

- Sending read request
- Reading the touch input data
- Access to the touch input data

The driver initialization routines allow the application to initialize the driver. The driver must be initialized before it can be used by application. Once the driver is initialized the driver open routine allows retrieving the client handle. Once the touch input is available a touch input read request is sent and input data is retrieved in a buffer. The buffer data is then decoded to get the x and y coordinate of the touch screen in the form of the number of pixels.

mXT336T Driver Abstraction Model



Library Overview

This section contains information about how the Touch Driver operates in a system.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the mXT336T Touch Driver.

Library Interface Section	Description
Device-specific Functions	Provides mXT336T-specific system module interfaces, device initialization, deinitialization, open, close, task, and status functions.
Generic Functions	Provides generic system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

This section describes the workings of the Touch Driver Library.

Description

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, open, close, task, and status functions.
- Read Request function, which provides Touch input data read request function
- Read Touch Input function, which provides functions retrieving updated Touch input in the form x and y coordinates.

Initializing the Driver

Before the mXT336T Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the `DRV_TOUCH_INIT` data structure that is passed to the `DRV_MXT336T_Initialize` and the `DRV_MXT_Initialize` functions. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the mXT336T Touch Driver.

Example:

```
/* The following code shows an example of designing the
* DRV_TOUCH_INIT data structure. It also shows how an example
* usage of the DRV_TOUCH_MXT336T_Initialize function.
```

```
*/
DRV_TOUCH_INIT drvTouchInitData;
DRV_MXT_INIT drvMxtInitData;
SYS_MODULE_OBJ objectHandle;

const DRV_MXT336T_INIT drvTouchInitData =
{
 .moduleInit = {0},
 .touchId = DRV_TOUCH_INDEX_0,
 .drvInitialize = NULL,
 .drvOpen = DRV_I2C_Open,
 .orientation = 0,
 .horizontalResolution = 480,
 .verticalResolution = 272,
 .interruptSource = INT_SOURCE_EXTERNAL_1,
 .interruptChannel = PORT_CHANNEL_E,
 .interruptPin = PORTS_BIT_POS_8,
 .resetChannel = PORT_CHANNEL_A,
 .resetPin = PORTS_BIT_POS_2,
};

const DRV_MXT_INIT drvMxtInitData =
{
 .moduleInit = {0},
 .mxtId = DRV_MXT_INDEX_0,
 .drvInitialize = NULL,
 .orientation = 0,
 .horizontalResolution = 480,
 .verticalResolution = 272,
};

/* Driver initialization */
 sysObj.drvMXT336T = DRV_MXT336T_Initialize(DRV_MXT336T_INDEX_0,
 (SYS_MODULE_INIT *)&drvTouchInitData);

 sysObj.drvMxt0 = DRV_MXT_Initialize(DRV_MXT_INDEX_0,
 (SYS_MODULE_INIT *)&drvMxtInitData);
```

Opening the Driver

To use the mXT336T Touch Driver, the application must open the driver. This is done by calling the DRV_MXT_Open function.

If successful, the DRV_MXT_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_MXT_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

The open function in the driver is called from the system initialization routine by assigning a function pointer from sys_init object.

```
const DRV_TOUCH_INIT sysTouchInit0 =
{
 .drvInitialize = DRV_MXT_Initialize,
 .drvOpen = DRV_MXT_Open,
.
.

};

SYS_TOUCH_HANDLE SYS_TOUCH_Open
(
 const SYS_MODULE_INDEX moduleIndex
)
{
 SYS_TOUCH_CLIENT_OBJ *clientObj;
 SYS_TOUCH_OBJ *dObj;
 .
 .
 .

 /* open touch driver */
 dObj->driverInitData->drvOpen(moduleIndex, DRV_IO_INTENT_READWRITE);
```

```
}
```

Touch Input Read Request

To read the touch input from the mXT336T touch controller device, a read request must be registered. This is done by calling DRV_MXT336T_ReadRequest. If successful, it registers a buffer read request to the I2C command queue. It also adds an input decode command to the mXT336T command queue once the I2C returns with touch input data. It can return error if the driver instance object is invalid or the mXT336T command queue is full. The read request is to be called from the mXT336T ISR. This ISR is triggered once the touch input is available. The following code shows an example of a mXT336T read request registration:

```
SYS_MODULE_OBJ object; // Returned from DRV_TOUCH_MXT336T_Initialize

void ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMxt336t(void)
{
DRV__MXT336T_ReadRequest ( object );

// Do other tasks
```

Tasks Routine

This routine processes the mXT336T commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The mXT336T Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
 SYS_MODULE_OBJ drvMXT336T;
 SYS_MODULE_OBJ drvMxt0;; // Returned from DRV_TOUCH_MXT336T_Initialize

void SYS_Tasks( void )
{
 DRV_MXT336T_Tasks(sysObj.drvMXT336T);
 DRV_MXT_Tasks(sysObj.drvMxt0);

// Do other tasks
}
```

Configuring the Library

Macros

	Name	Description
	DRV_MXT336T_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_MXT336T_CALIBRATION_INSET	Defines the calibration inset.
	DRV_MXT336T_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_MXT336T_INDEX	MXT336T static index selection.
	DRV_MXT336T_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_MXT336T_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_MXT336T_SAMPLE_POINTS	Define the sample points.
	DRV_MXT336T_TOUCH_DIAMETER	Defines the touch diameter.

Description

The configuration of the mXT336T Touch Driver is based on the file system_config.h.

This header file contains the configuration selection for the mXT336T Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the mXT336T Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_MXT336T_CALIBRATION_DELAY Macro

Defines the calibration delay.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_CALIBRATION_DELAY 300
```

Description

MXT336T Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_MXT336T_CALIBRATION_INSET Macro

Defines the calibration inset.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_CALIBRATION_INSET 25
```

Description

MXT336T Calibration Inset

This macro defines the calibration inset.

Remarks

None.

DRV_MXT336T_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_CLIENTS_NUMBER 5
```

Description

MXT336T maximum number of clients

This macro selects the maximum number of clients.

This definition selected the maximum number of clients that the MXT336T driver can support at run time.

Remarks

None.

DRV_MXT336T_INDEX Macro

MXT336T static index selection.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_INDEX DRV_MXT336T_INDEX_0
```

Description

MXT336T Static Index Selection

This macro specifies the static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_MXT336T_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_INSTANCES_NUMBER 1
```

Description

MXT336T hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_MXT336T_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_INTERRUPT_MODE false
```

Description

MXT336T Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of MXT336T operation is desired
- false - Select if polling mode of MXT336T operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_MXT336T_SAMPLE_POINTS Macro

Define the sample points.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_SAMPLE_POINTS 4
```

Description

MXT336T Sample Points

MXT336T sample points

Remarks

None.

DRV_MXT336T_TOUCH_DIAMETER Macro

Defines the touch diameter.

File

drv_mxt336t_config_template.h

C

```
#define DRV_MXT336T_TOUCH_DIAMETER 10
```

Description

MXT336T Touch Diameter

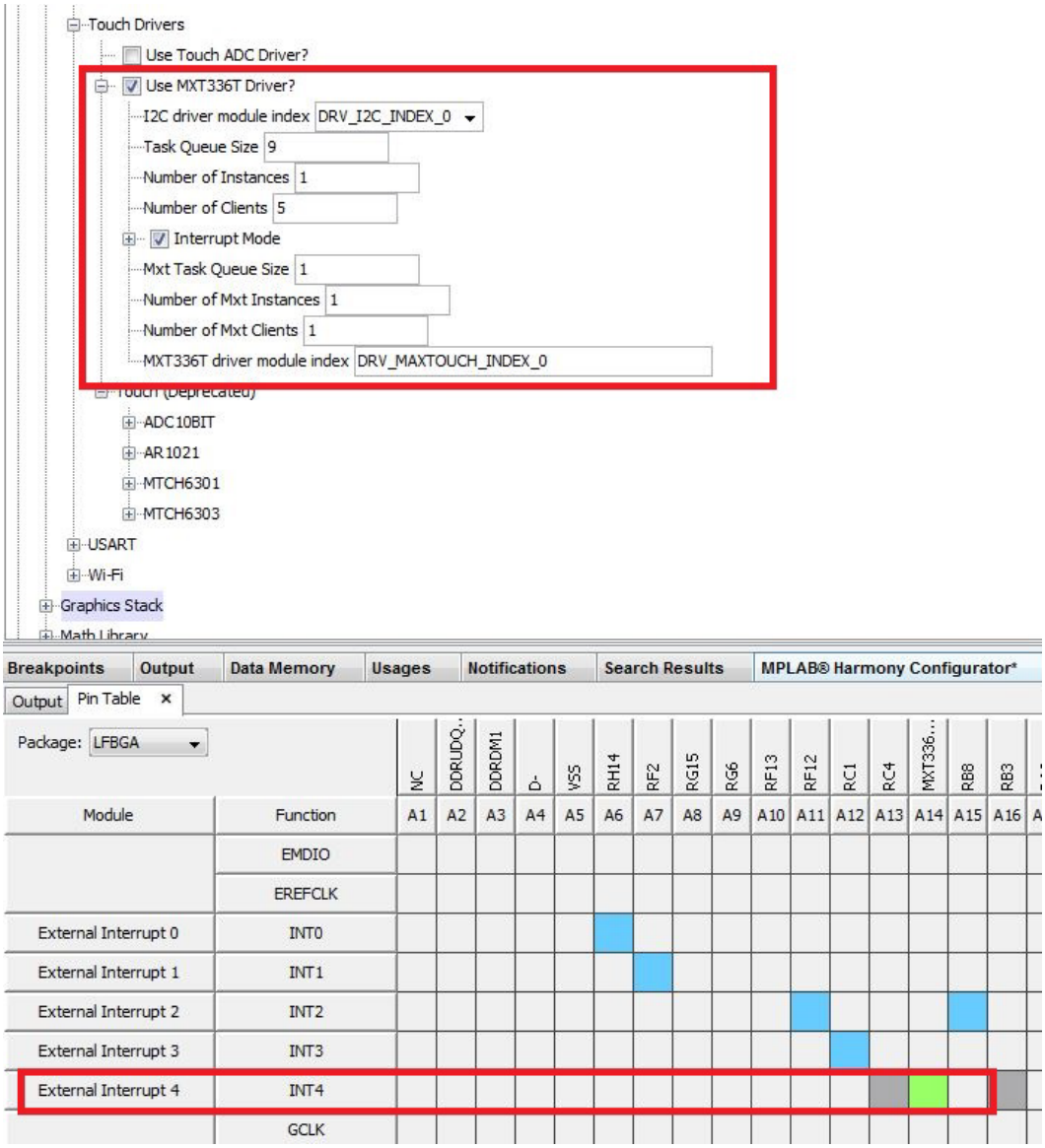
This macro defines the touch diameter

Remarks

None.

Configuring the MHC

The following figure details the settings required to configure the MHC for the mXT336T Touch Driver.



Building the Library

This section lists the files that are available in the mXT336T Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the mXT336T Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/mxt336t`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_mxt336t.h	Header file that exports the device-specific driver API.
/drv_mxt.h	Header file for generic driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
src/drv_mxt336t.c	Basic mXT336T Touch Driver implementation file.
src/drv_mxt.c	Generic maXTouch touch driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The mXT336T Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) Device-specific Functions

	Name	Description
	DRV_MXT336T_Close	Closes an opened instance of the MXT336T driver. **Implementation:** Dynamic
	DRV_MXT336T_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_MXT336T_Open	Opens the specified MXT336T driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT336T_CloseObject	Closes an opened instance of the MXT336T client object
	DRV_MXT336T_OpenObject	Opens the specified MXT336T object driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet	Sets the event handler for a MXT336T client object
	DRV_MXT336T_Deinitialize	Deinitializes the specified instance of the MXT336T driver module. **Implementation:** Dynamic
	DRV_MXT336T_Initialize	Initializes the MXT336T instance for the specified driver index
	DRV_MXT336T_Status	Provides the current status of the MXT336T driver module. **Implementation:** Dynamic

	DRV_MXT336T_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic

b) Generic Functions

	Name	Description
	DRV_MXT_Close	Closes an opened instance of the MXT driver. **Implementation:** Dynamic
	DRV_MXT_MaxtouchEventCallback	
	DRV_MXT_Deinitialize	Deinitializes the specified instance of the MXT driver module. **Implementation:** Dynamic
	DRV_MXT_Open	Opens the specified MXT driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_MXT_Initialize	Initializes the MXT instance for the specified driver index. **Implementation:** Dynamic
	DRV_MXT_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_MXT_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_MXT_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_MXT_Status	Provides the current status of the MXT driver module. **Implementation:** Dynamic
	DRV_MXT_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_MXT_TouchStatus	Returns the status of the current touch input.

c) Data Types and Constants

	Name	Description
	_DRV_MXT_CLIENT_OBJECT	MXT Driver client object maintaining client data.
	DRV_MXT_CLIENT_OBJECT	MXT Driver client object maintaining client data.
	DRV_MXT_HANDLE	Touch screen controller MXT driver handle.
	DRV_MXT_INIT	Defines the data required to initialize or reinitialize the MXT driver
	DRV_MXT_MODULE_ID	Number of valid MXT driver indices.
	DRV_MXT_OBJECT	Defines the data structure maintaining MXT driver instance object.
	DRV_MXT_TASK_QUEUE	Defines the MXT Touch Controller driver task data structure.
	DRV_MXT_TASK_STATE	Enumeration defining MXT touch controller driver task state.
	DRV_MXT336T_CLIENT_CALLBACK	Pointer to a MXT336T client callback function data type.
	DRV_MXT336T_HANDLE	Touch screen controller MXT336T driver handle.
	DRV_MXT336T_INIT	Defines the data required to initialize or reinitialize the MXT336T driver
	DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA	This structure maintains the information associated with each msg received or event that occurs
	DRV_MXT336T_OBJECT_TYPE	The enum lists the different objects supported by the maxtouch device.
	DRV_MXT_HANDLE_INVALID	Definition of an invalid handle.
	_DRV_MXT336T_H	This is macro _DRV_MXT336T_H.
	DRV_MXT_I2C_MASTER_READ_ID	MXT input read, I2C address from where master reads touch input data.
	DRV_MXT_I2C_MASTER_WRITE_ID	MXT command register write, I2C address where master sends the commands.
	DRV_MXT_I2C_READ_FRAME_SIZE	I2C Frame size for reading MXT touch input.
	DRV_MXT_INDEX_0	MXT driver index definitions.
	DRV_MXT_INDEX_1	This is macro DRV_MXT_INDEX_1.
	DRV_MXT_INDEX_COUNT	Number of valid Touch controller MXT driver indices.
	DRV_MXT336T_HANDLE_INVALID	Definition of an invalid handle.
	DRV_MXT336T_I2C_FRAME_SIZE	I2C Frame size for reading MXT336T touch input.
	DRV_MXT336T_I2C_MASTER_READ_ID	MXT336T input read, I2C address from where master reads touch input data.

	DRV_MXT336T_I2C_MASTER_WRITE_ID	MXT336T command register write, I2C address where master sends the commands.
	DRV_MXT336T_I2C_READ_ID_FRAME_SIZE	This is macro DRV_MXT336T_I2C_READ_ID_FRAME_SIZE.
	DRV_MXT336T_INDEX_0	MXT336T driver index definitions.
	DRV_MXT336T_INDEX_1	This is macro DRV_MXT336T_INDEX_1.
	DRV_MXT336T_INDEX_COUNT	Number of valid Touch controller MXT336T driver indices.
	t100_event	Types of touch events reported by the Maxtouch Multi touch object
	t100_type	Types of touch types reported by the Maxtouch Multi touch object
	DRV_MXT336T_T100_XRANGE	MXT336T Driver Object Register Adresses for the registers being read in the driver
	DRV_MXT336T_T100_YRANGE	This is macro DRV_MXT336T_T100_YRANGE.

Description

This section describes the functions of the mXT336T Touch Driver Library.

Refer to each section for a detailed description.

a) Device-specific Functions

DRV_MXT336T_Close Function

Closes an opened instance of the MXT336T driver.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
void DRV_MXT336T_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MXT336T driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_MXT336T_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_MXT336T_Initialize routine must have been called for the specified MXT336T driver instance.

DRV_MXT336T_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_MXT336T_Open

DRV_MXT336T_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_MXT336T_Close (DRV_HANDLE handle)

DRV_MXT336T_ReadRequest Function

Sends a read request to I2C bus driver and adds the read task to queue.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
void DRV_MXT336T_ReadRequest(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to send a touch input read request to the I2C bus driver and adding the input read decode task to the queue. It is always called from MXT336T interrupt ISR routine.

Remarks

This function is normally not called directly by an application. It is called by the MXT336T ISR routine.

Preconditions

The DRV_MXT336T_Initialize routine must have been called for the specified MXT336T driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_MXT336T_Initialize

void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMXT(void)
{
    DRV_MXT336T_ReadRequest ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_MXT336T_Initialize)

Function

void DRV_MXT336T_ReadRequest(SYS_MODULE_OBJ object)

DRV_MXT336T_Open Function

Opens the specified MXT336T driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
DRV_HANDLE DRV_MXT336T_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_MXT336T_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified MXT336T driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the MXT336T device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_MXT336T_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_MXT336T_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_MXT336T_Open( DRV_MXT336T_INDEX_0,
                                  DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_MXT336T_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_MXT336T_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_MXT336T_CloseObject Function

Closes an opened instance of the MXT336T client object

File

drv_mxt336t.h

C

```
void DRV_MXT336T_CloseObject(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MXT336T client object, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_MXT336T_OpenObject before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_MXT336T_Initialize routine must have been called for the specified MXT336T driver instance.

DRV_MXT336T_OpenObject must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_MXT336T_Open

DRV_MXT336T_CloseObject ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_MXT336T_CloseObject (DRV_HANDLE handle)

DRV_MXT336T_OpenObject Function

Opens the specified MXT336T object driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
DRV_HANDLE DRV_MXT336T_OpenObject(const DRV_HANDLE deviceHandle, const uint8_t objType, const uint8_t
objInstance);
```

Returns

If successful, the routine returns a valid object-instance handle (

Description

This routine opens the specified MXT336T object driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

The DRV_MXT336T_Initialize function must have been called before calling this function. The driver must have been opened.

Example

```
DRV_HANDLE  handle;

handle = DRV_MXT336T_OpenObject(drvHandle, GEN_PROCESSOR_T5, 1);
```

Parameters

Parameters	Description
deviceHandle	Handle of the MXT336T device
objType	Object type being requested
objInstance	Instance of the object of this type

Function

DRV_HANDLE DRV_MXT336T_OpenObject (const DRV_HANDLE deviceHandle, const uint8_t objType,
const uint8_t objInstance)

DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet Function

Sets the event handler for a MXT336T client object

File

drv_mxt336t.h

C

```
bool DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet(const DRV_HANDLE clientHandle, const
DRV_MXT336T_CLIENT_CALLBACK callback, uintptr_t context);
```

Returns

bool - true if the handler was successfully set

- false if the handler could not be set

Description

This function sets the event handler used to handle report messages from a MXT336T object.

Preconditions

The DRV_MXT336T_OpenObject routine must have been called for the specified MXT336T driver instance.

DRV_MXT336T_OpenObject must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_MXT336T_OpenObject

DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet(handle, objectCallback, NULL);
```

Parameters

Parameters	Description
clientHandle	A valid open-instance handle, returned from the driver's openobject routine
callback	A callback function to handle report messages
context	The context for the call

Function

bool DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet(const DRV_HANDLE clientHandle,

const DRV_MXT336T_CLIENT_CALLBACK callback, uintptr_t context)

DRV_MXT336T_Deinitialize Function

Deinitializes the specified instance of the MXT336T driver module.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
void DRV_MXT336T_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the MXT336T driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_MXT336T_Status operation. The system has to use DRV_MXT336T_Status to determine when the module is in the ready state.

Preconditions

Function DRV_MXT336T_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Parameter: object - Driver object handle, returned from DRV_MXT336T_Initialize

Example

```
SYS_MODULE_OBJ      object;    //Returned from DRV_MXT336T_Initialize
SYS_STATUS          status;

DRV_MXT336T_Deinitialize ( object );

status = DRV_MXT336T_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_MXT336T_Deinitialize (SYS_MODULE_OBJ object)

DRV_MXT336T_Initialize Function

Initializes the MXT336T instance for the specified driver index

File

drv_mxt336t.h

C

```
SYS_MODULE_OBJ DRV_MXT336T_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MXT336T driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the MXT336T module ID. For example, driver instance 0 can be assigned to MXT336T2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_MXT336T_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other MXT336T routine is called.

This routine should only be called once during system initialization unless DRV_MXT336T_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_MXT336T_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the MXT336T initialization structure
// Touch Module Id
init.moduleInit                 = {0},
init.touchId                    = DRV_TOUCH_INDEX_0,
init.drvInitialize              = NULL,
init.drvOpen                    = DRV_I2C_Open,
init.interruptSource            = INT_SOURCE_EXTERNAL_1,
init.interruptChannel           = PORT_CHANNEL_D,
init.interruptPin               = PORTS_BIT_POS_1,
init.resetChannel               = PORT_CHANNEL_A,
init.resetPin                   = PORTS_BIT_POS_14,

objectHandle = DRV_MXT336T_Initialize(DRV_TOUCH_INDEX_0,
                                      (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the MXT336T ID. The hardware MXT336T ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_MXT336T_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_MXT336T_Status Function

Provides the current status of the MXT336T driver module.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
SYS_STATUS DRV_MXT336T_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system-level operation and cannot start another

Description

This function provides the current status of the MXT336T driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_MXT336T_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;  // Returned from DRV_MXT336T_Initialize
SYS_STATUS          status;

status = DRV_MXT336T_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_MXT336T_Initialize

Function

SYS_STATUS DRV_MXT336T_Status (SYS_MODULE_OBJ object)

DRV_MXT336T_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_mxt336t.h

C

```
void DRV_MXT336T_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function. This routine decodes the touch input data available in drvI2CReadFrameData.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_MXT336T_Initialize routine must have been called for the specified MXT336T driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_MXT336T_Initialize

void SYS_Tasks( void )
{
    DRV_MXT336T_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_MXT336T_Initialize)

Function

void DRV_MXT336T_Tasks (SYS_MODULE_OBJ object);

b) Generic Functions

DRV_MXT_Close Function

Closes an opened instance of the MXT driver.

Implementation: Dynamic

File

drv_mxt.h

C

```
void DRV_MXT_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MXT driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_MXT_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_MXT_Initialize routine must have been called for the specified MXT driver instance.

DRV_MXT_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_MXT_Open
```

```
DRV_MXT_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_MXT_Close (DRV_HANDLE handle)

DRV_MXT_MaxtouchEventCallback Function

File

drv_mxt.h

C

```
void DRV_MXT_MaxtouchEventCallback(DRV_HANDLE clientObject, DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA *
updateObject, uintptr_t context);
```

Remarks

See prototype in app.h.

Function

void DRV_MXT_MaxtouchEventCallback (DRV_HANDLE clientObject,

DRV_MAXTOUCH_OBJECT_CLIENT_EVENT_DATA *updateObject, uintptr_t context);

DRV_MXT_Deinitialize Function

Deinitializes the specified instance of the MXT driver module.

Implementation: Dynamic

File

drv_mxt.h

C

```
void DRV_MXT_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the MXT driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_MXT_Status operation. The system has to use DRV_MXT_Status to determine when the module is in the ready state.

Preconditions

Function DRV_MXT_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Parameter: object - Driver object handle, returned from DRV_MXT_Initialize

Example

```
SYS_MODULE_OBJ      object;    //Returned from DRV_MXT_Initialize
SYS_STATUS          status;

DRV_MXT_Deinitialize ( object );

status = DRV_MXT_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
```

```
    // when the driver is deinitialized.
}
```

Function

void DRV_MXT_Deinitialize (SYS_MODULE_OBJ object)

DRV_MXT_Open Function

Opens the specified MXT driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_mxt.h

C

```
DRV_HANDLE DRV_MXT_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_MXT_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified MXT driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the MXT device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_MXT_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_MXT_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_MXT_Open( DRV_MXT_INDEX_0,
                                    DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_MXT_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_MXT_Open (const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT intent)

DRV_MXT_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_mxt.h

C

```
void DRV_MXT_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_MXT_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_MXT_Initialize Function

Initializes the MXT instance for the specified driver index.

Implementation: Dynamic

File

drv_mxt.h

C

```
SYS_MODULE_OBJ DRV_MXT_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MXT driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the MXT module ID. For example, driver instance 0 can be assigned to MXT2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_MXT_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other MXT routine is called.

This routine should only be called once during system initialization unless DRV_MXT_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_MXT_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the MXT initialization structure
// Touch Module Id
init.touchId                    = DRV_TOUCH_INDEX_0;

// I2C Bus driver open
init.drvOpen                    = DRV_I2C_Open;

// Interrupt Source for Touch
init.interruptSource            = INT_SOURCE_EXTERNAL_1;

// Interrupt Pin function mapping
```

```
init.interruptPort.inputFunction  = INPUT_FUNC_INT1;

// Pin to be mapped as interrupt pin
init.interruptPort.inputPin       = INPUT_PIN_RPE8;

// Analog pin number
init.interruptPort.analogPin      = PORTS_ANALOG_PIN_25;

// Pin Mode of analog pin
init.interruptPort.pinMode        = PORTS_PIN_MODE_DIGITAL;

// Interrupt pin port
init.interruptPort.channel        = PORT_CHANNEL_E;

// Interrupt pin port maskl
init.interruptPort.dataMask       = 0x8;

// Touch screen orientation
init.orientation                  = DISP_ORIENTATION;

// Touch screen horizontal resolution
init.horizontalResolution         = DISP_HOR_RESOLUTION;

// Touch screen vertical resolution
init.verticalResolution           = DISP_VER_RESOLUTION;


objectHandle = DRV_MXT_Initialize(DRV_TOUCH_INDEX_0,
                                  (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the MXT ID. The hardware MXT ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_MXT_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

DRV_MXT_ReadRequest Function

Sends a read request to I2C bus driver and adds the read task to queue.

Implementation: Dynamic

File

drv_mxt.h

C

```
void DRV_MXT_ReadRequest(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to send a touch input read request to the I2C bus driver and adding the input read decode task to the queue. It is always called from MXT interrupt ISR routine.

Remarks

This function is normally not called directly by an application. It is called by the MXT ISR routine.

Preconditions

The DRV_MXT_Initialize routine must have been called for the specified MXT driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_MXT_Initialize

void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMXT(void)
{
    DRV_MXT_ReadRequest ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_MXT_Initialize)

Function

void DRV_MXT_ReadRequest(SYS_MODULE_OBJ object)

DRV_MXT_TouchGetX Function

Returns the x coordinate of touch input.

Implementation: Dynamic

File

drv_mxt.h

C

```
short DRV_MXT_TouchGetX(uint8_t touchNumber);
```

Returns

It returns the x coordinate of the touch input in terms of number of pixels.

Description

It returns the x coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_MXT_TouchGetX(uint8 touchNumber)

DRV_MXT_TouchGetY Function

Returns the y coordinate of touch input.

Implementation: Dynamic

File

drv_mxt.h

C

```
short DRV_MXT_TouchGetY(uint8_t touchNumber);
```

Returns

It returns the y coordinate of the touch input in terms of number of pixels.

Description

It returns the y coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_MXT_TouchGetY(uint8 touchNumber)

DRV_MXT_Status Function

Provides the current status of the MXT driver module.

Implementation: Dynamic

File

drv_mxt.h

C

```
SYS_STATUS DRV_MXT_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system-level operation and cannot start another

Description

This function provides the current status of the MXT driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_MXT_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;  // Returned from DRV_MXT_Initialize
SYS_STATUS          status;

status = DRV_MXT_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_MXT_Initialize

Function

SYS_STATUS DRV_MXT_Status (SYS_MODULE_OBJ object)

DRV_MXT_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_mxt.h

C

```
void DRV_MXT_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function. This routine decodes the touch input data available in drvI2CReadFrameData.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_MXT_Initialize routine must have been called for the specified MXT driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_MXT_Initialize

void SYS_Tasks( void )
{
    DRV_MXT_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_MXT_Initialize)

Function

void DRV_MXT_Tasks (SYS_MODULE_OBJ object);

DRV_MXT_TouchStatus Function

Returns the status of the current touch input.

File

drv_mxt.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_MXT_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_MXT_TouchStatus(const SYS_MODULE_INDEX index)

c) Data Types and Constants

DRV_MXT_CLIENT_OBJECT Structure

MXT Driver client object maintaining client data.

File

drv_mxt.h

C

```
typedef struct _DRV_MXT_CLIENT_OBJECT {
  DRV_MXT_OBJECT* driverObject;
  DRV_IO_INTENT intent;
} DRV_MXT_CLIENT_OBJECT;
```

Members

Members	Description
DRV_MXT_OBJECT* driverObject;	Driver Object associated with the client
DRV_IO_INTENT intent;	The intent with which the client was opened

Description

Structure DRV_MXT_CLIENT_OBJECT

This defines the object required for the maintenance of the software clients instance. This object exists once per client instance.

Remarks

None.

DRV_MXT_HANDLE Type

Touch screen controller MXT driver handle.

File

drv_mxt.h

C

```
typedef uintptr_t DRV_MXT_HANDLE;
```

Description

MXT Driver Handle

Touch controller MXT driver handle is a handle for the driver client object. Each driver with succesful open call will return a new handle to the client object.

Remarks

None.

DRV_MXT_INIT Structure

Defines the data required to initialize or reinitialize the MXT driver

File

drv_mxt.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  int mxtId;
  SYS_MODULE_OBJ (* drvInitialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
  SYS_MODULE_INDEX maxtouchID;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
} DRV_MXT_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
int mxtId;	ID
SYS_MODULE_OBJ (* drvInitialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);	initialize function for module (normally called statically
SYS_MODULE_INDEX maxtouchID;	index for the maxtouch driver instance used by this driver
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels

Description

Structure DRV_MXT_INIT

This data type defines the data required to initialize or reinitialize the MXT driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_MXT_MODULE_ID Enumeration

Number of valid MXT driver indices.

File

drv_mxt.h

C

```
typedef enum {
  MXT_ID_1 = 0,
  MXT_NUMBER_OF_MODULES
} DRV_MXT_MODULE_ID;
```

Description

Enumeration: DRV_MXT_MODULE_ID

This constant identifies the number of valid MXT driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_MXT_OBJECT Structure

Defines the data structure maintaining MXT driver instance object.

File

drv_mxt.h

C

```
typedef struct {
  SYS_STATUS status;
  int mxtId;
  SYS_MODULE_INDEX drvIndex;
  bool inUse;
  bool isExclusive;
  uint8_t numClients;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
  int32_t readRequest;
  SYS_MODULE_INDEX maxtouchID;
  DRV_HANDLE hMaxtouch;
  DRV_HANDLE hMaxtouchGestureClient;
  DRV_TOUCH_POSITION_STATUS touchStatus;
```

```
    bool maxtouchDataAvailable;
    uint8_t maxtouchData[32];
    uint16_t xRange;
    uint16_t yRange;
} DRV_MXT_OBJECT;
```

Members

Members	Description
SYS_STATUS status;	The status of the driver
int mxtId;	The peripheral Id associated with the object
SYS_MODULE_INDEX drvIndex;	Save the index of the driver. Important to know this as we are using reference based accessing
bool inUse;	Flag to indicate instance in use
bool isExclusive;	Flag to indicate module used in exclusive access mode
uint8_t numClients;	Number of clients possible with the hardware instance
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels
uint16_t verticalResolution;	Vertical Resolution of the displayed orientaion in Pixels
int32_t readRequest;	Touch Input read request counter
SYS_MODULE_INDEX maxtouchID;	index of the maxtouch driver being used
DRV_HANDLE hMaxtouch;	handle for the maxtouch driver being used
DRV_HANDLE hMaxtouchGestureClient;	handle for the maxtouch driver object we are listening to
DRV_TOUCH_POSITION_STATUS touchStatus;	Touch status
bool maxtouchDataAvailable;	flag to indicate new maxtouch data is available
uint8_t maxtouchData[32];	data from the maxtouch device

Description

Structure DRV_MXT_OBJECT

This data structure maintains the MXT driver instance object. The object exists once per hardware instance.

Remarks

None.

DRV_MXT_TASK_QUEUE Structure

Defines the MXT Touch Controller driver task data structure.

File

drv_mxt.h

C

```
typedef struct {
  bool inUse;
  DRV_MXT_TASK_STATE taskState;
  DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;
  uint8_t drvI2CReadFrameData[DRV_MXT_I2C_READ_FRAME_SIZE];
} DRV_MXT_TASK_QUEUE;
```

Members

Members	Description
bool inUse;	Flag denoting the allocation of task
DRV_MXT_TASK_STATE taskState;	Enum maintaining the task state
DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;	I2C Buffer handle

uint8_t drvI2CReadFrameData[DRV_MXT_I2C_READ_FRAME_SIZE];	Response to Read Touch Input Command • Response = { MXT Read Address, • Input Data Size, • Touch Id, Pen status, • Touch X coordinate (0 to 6), • Touch X coordinate (7 to 11), • Touch Y coordinate (0 to 6), • Touch Y coordinate (7 to 11) }

Description

Structure DRV_MXT_TASK_QUEUE

This data type defines the data structure maintaing task context in the task queue. The inUse flag denotes the task context allocation for a task. The enum variable taskState maintains the current task state. The I2C buffer handle drvI2CReadBufferHandle maintains the I2C driver buffer handle returned by the I2C driver read request. The byte array variable drvI2CReadFrameData maintains the I2C frame data sent by MXT after a successful read request.

Remarks

None.

DRV_MXT_TASK_STATE Enumeration

Enumeration defining MXT touch controller driver task state.

File

drv_mxt.h

C

```
typedef enum {
  DRV_MXT_TASK_STATE_INIT = 0,
  DRV_MXT_TASK_STATE_READ_INPUT,
  DRV_MXT_TASK_STATE_DECODE_INPUT,
  DRV_MXT_TASK_STATE_DONE
} DRV_MXT_TASK_STATE;
```

Members

Members	Description
DRV_MXT_TASK_STATE_INIT = 0	Task initialize state
DRV_MXT_TASK_STATE_READ_INPUT	Task read touch input request state
DRV_MXT_TASK_STATE_DECODE_INPUT	Task touch input decode state
DRV_MXT_TASK_STATE_DONE	Task complete state

Description

Enumeration DRV_MXT_TASK_STATE

This enumeration defines the MXT touch controller driver task state. The task state helps to synchronize the operations of initialization the the task, adding the read input task to the task queue once the touch controller notifies the available touch input and a decoding the touch input received.

Remarks

None.

DRV_MXT336T_CLIENT_CALLBACK Type

Pointer to a MXT336T client callback function data type.

File

drv_mxt336t.h

C

```
typedef void (* DRV_MXT336T_CLIENT_CALLBACK)(DRV_HANDLE clientObject, DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA
*updateObject, uintptr_t context);
```

Description

MXT336T Driver Callback Function Pointer

This data type defines a pointer to a MXT336T client callback function.

DRV_MXT336T_HANDLE Type

Touch screen controller MXT336T driver handle.

File

drv_mxt336t.h

C

```
typedef uintptr_t DRV_MXT336T_HANDLE;
```

Description

MXT336T Driver Handle

Touch controller MXT336T driver handle is a handle for the driver client object. Each driver with successful open call will return a new handle to the client object.

Remarks

None.

DRV_MXT336T_INIT Type

Defines the data required to initialize or reinitialize the MXT336T driver

File

drv_mxt336t.h

C

```
typedef struct DRV_MXT336T_INIT@2 DRV_MXT336T_INIT;
```

Description

Structure DRV_MXT336T_INIT

This data type defines the data required to initialize or reinitialize the MXT336T driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA Structure

This structure maintains the information associated with each msg received or event that occurs

File

drv_mxt336t.h

C

```
typedef struct {
  uint8_t reportID;
  uint8_t dataSize;
  uint8_t * pData;
  uint16_t xRange;
  uint16_t yRange;
} DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA;
```

Members

Members	Description
uint8_t reportID;	report ID within the object
uint8_t dataSize;	max size of data
uint8_t * pData;	data associated with the report

Description

Structure DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA

This structure maintains the information associated with each msg received or event that occurs. Each msg gets a reportID that identifies the object reporting the change in status. For touch messages the xRange and yRange for the touch device gets reported and the data pointer contains the status msg information which has the touch type, touch event and touch coordinates.

Remarks

None.

DRV_MXT336T_OBJECT_TYPE Enumeration

The enum lists the different objects supported by the maxtouch device.

File

drv_mxt336t.h

C

```
typedef enum {
  DRV_MXT336T_OBJECT_RESERVED_T0 = 0,
  DRV_MXT336T_OBJECT_RESERVED_T1 = 1,
  DRV_MXT336T_OBJECT_DEBUG_DELTAS_T2 = 2,
  DRV_MXT336T_OBJECT_DEBUG_REFERENCES_T3 = 3,
  DRV_MXT336T_OBJECT_DEBUG_SIGNALS_T4 = 4,
  DRV_MXT336T_OBJECT_GEN_MESSAGEPROCESSOR_T5 = 5,
  DRV_MXT336T_OBJECT_GEN_COMMANDPROCESSOR_T6 = 6,
  DRV_MXT336T_OBJECT_GEN_POWERCONFIG_T7 = 7,
  DRV_MXT336T_OBJECT_GEN_ACQUISITIONCONFIG_T8 = 8,
  DRV_MXT336T_OBJECT_TOUCH_MULTITOUCHSCREEN_T9 = 9,
  DRV_MXT336T_OBJECT_TOUCH_SINGLETOUCHSCREEN_T10 = 10,
  DRV_MXT336T_OBJECT_TOUCH_XSLIDER_T11 = 11,
  DRV_MXT336T_OBJECT_TOUCH_YSLIDER_T12 = 12,
  DRV_MXT336T_OBJECT_TOUCH_XWHEEL_T13 = 13,
  DRV_MXT336T_OBJECT_TOUCH_YWHEEL_T14 = 14,
  DRV_MXT336T_OBJECT_TOUCH_KEYARRAY_T15 = 15,
  DRV_MXT336T_OBJECT_PROCG_SIGNALFILTER_T16 = 16,
  DRV_MXT336T_OBJECT_PROCI_LINEARIZATIONTABLE_T17 = 17,
  DRV_MXT336T_OBJECT_SPT_COMMSCONFIG_T18 = 18,
  DRV_MXT336T_OBJECT_SPT_GPIOPWM_T19 = 19,
  DRV_MXT336T_OBJECT_PROCI_GRIPFACESUPPRESSION_T20 = 20,
  DRV_MXT336T_OBJECT_RESERVED_T21 = 21,
  DRV_MXT336T_OBJECT_PROCG_NOISESUPPRESSION_T22 = 22,
  DRV_MXT336T_OBJECT_TOUCH_PROXIMITY_T23 = 23,
  DRV_MXT336T_OBJECT_PROCI_ONETOUCHGESTUREPROCESSOR_T24 = 24,
  DRV_MXT336T_OBJECT_SPT_SELFTEST_T25 = 25,
  DRV_MXT336T_OBJECT_DEBUG_CTERANGE_T26 = 26,
  DRV_MXT336T_OBJECT_PROCI_TWOTOUCHGESTUREPROCESSOR_T27 = 27,
  DRV_MXT336T_OBJECT_SPT_CTECONFIG_T28 = 28,
  DRV_MXT336T_OBJECT_SPT_GPI_T29 = 29,
  DRV_MXT336T_OBJECT_SPT_GATE_T30 = 30,
  DRV_MXT336T_OBJECT_TOUCH_KEYSET_T31 = 31,
  DRV_MXT336T_OBJECT_TOUCH_XSLIDERSET_T32 = 32,
  DRV_MXT336T_OBJECT_RESERVED_T33 = 33,
  DRV_MXT336T_OBJECT_GEN_MESSAGEBLOCK_T34 = 34,
  DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T35 = 35,
  DRV_MXT336T_OBJECT_RESERVED_T36 = 36,
  DRV_MXT336T_OBJECT_DEBUG_DIAGNOSTIC_T37 = 37,
  DRV_MXT336T_OBJECT_SPT_USERDATA_T38 = 38,
  DRV_MXT336T_OBJECT_SPARE_T39 = 39,
  DRV_MXT336T_OBJECT_PROCI_GRIPSUPPRESSION_T40 = 40,
  DRV_MXT336T_OBJECT_PROCI_PALMSUPPRESSION_T41 = 41,
  DRV_MXT336T_OBJECT_PROCI_TOUCHSUPPRESSION_T42 = 42,
  DRV_MXT336T_OBJECT_SPT_DIGITIZER_T43 = 43,
  DRV_MXT336T_OBJECT_SPT_MESSAGECOUNT_T44 = 44,
  DRV_MXT336T_OBJECT_PROCI_VIRTUALKEY_T45 = 45,
  DRV_MXT336T_OBJECT_SPT_CTECONFIG_T46 = 46,
  DRV_MXT336T_OBJECT_PROCI_STYLUS_T47 = 47,
  DRV_MXT336T_OBJECT_PROCG_NOISESUPPRESSION_T48 = 48,
  DRV_MXT336T_OBJECT_GEN_DUALPULSE_T49 = 49,
  DRV_MXT336T_OBJECT_SPARE_T50 = 50,
  DRV_MXT336T_OBJECT_SPT_SONY_CUSTOM_T51 = 51,
  DRV_MXT336T_OBJECT_TOUCH_PROXKEY_T52 = 52,
  DRV_MXT336T_OBJECT_GEN_DATASOURCE_T53 = 53,
  DRV_MXT336T_OBJECT_PROCG_NOISESUPPRESSION_T54 = 54,
  DRV_MXT336T_OBJECT_PROCI_ADAPTIVETHRESHOLD_T55 = 55,
```

```
    DRV_MXT336T_OBJECT_PROCI_SHIELDLESS_T56 = 56,
    DRV_MXT336T_OBJECT_PROCI_EXTRATOUCHSCREENDATA_T57 = 57,
    DRV_MXT336T_OBJECT_SPT_EXTRANOISESUPCTRLS_T58 = 58,
    DRV_MXT336T_OBJECT_SPT_FASTDRIFT_T59 = 59,
    DRV_MXT336T_OBJECT_SPT_TIMER_T61 = 61,
    DRV_MXT336T_OBJECT_PROCG_NOISESUPPRESSION_T62 = 62,
    DRV_MXT336T_OBJECT_PROCI_ACTIVESTYLUS_T63 = 63,
    DRV_MXT336T_OBJECT_SPT_REFERENCERELOAD_T64 = 64,
    DRV_MXT336T_OBJECT_PROCI_LENSBENDING_T65 = 65,
    DRV_MXT336T_OBJECT_SPT_GOLDENREFERENCES_T66 = 66,
    DRV_MXT336T_OBJECT_PROCI_CUSTOMGESTUREPROCESSOR_T67 = 67,
    DRV_MXT336T_OBJECT_SERIAL_DATA_COMMAND_T68 = 68,
    DRV_MXT336T_OBJECT_PROCI_PALMGESTUREPROCESSOR_T69 = 69,
    DRV_MXT336T_OBJECT_SPT_DYNAMICCONFIGURATIONCONTROLLER_T70 = 70,
    DRV_MXT336T_OBJECT_SPT_DYNAMICCONFIGURATIONCONTAINER_T71 = 71,
    DRV_MXT336T_OBJECT_PROCG_NOISESUPPRESSION_T72 = 72,
    DRV_MXT336T_OBJECT_PROCI_ZONEINDICATION_T73 = 73,
    DRV_MXT336T_OBJECT_PROCG_SIMPLEGESTUREPROCESSOR_T74 = 74,
    DRV_MXT336T_OBJECT_MOTION_SENSING_OBJECT_T75 = 75,
    DRV_MXT336T_OBJECT_PROCI_MOTION_GESTURES_T76 = 76,
    DRV_MXT336T_OBJECT_SPT_CTESCANCONFIG_T77 = 77,
    DRV_MXT336T_OBJECT_PROCI_GLOVEDETECTION_T78 = 78,
    DRV_MXT336T_OBJECT_SPT_TOUCHEVENTTRIGGER_T79 = 79,
    DRV_MXT336T_OBJECT_PROCI_RETRANSMISSIONCOMPENSATION_T80 = 80,
    DRV_MXT336T_OBJECT_PROCI_UNLOCKGESTURE_T81 = 81,
    DRV_MXT336T_OBJECT_SPT_NOISESUPEXTENSION_T82 = 82,
    DRV_MXT336T_OBJECT_ENVIRO_LIGHTSENSING_T83 = 83,
    DRV_MXT336T_OBJECT_PROCI_GESTUREPROCESSOR_T84 = 84,
    DRV_MXT336T_OBJECT_PEN_ACTIVESTYLUSPOWER_T85 = 85,
    DRV_MXT336T_OBJECT_PROCG_NOISESUPACTIVESTYLUS_T86 = 86,
    DRV_MXT336T_OBJECT_PEN_ACTIVESTYLUSDATA_T87 = 87,
    DRV_MXT336T_OBJECT_PEN_ACTIVESTYLUSRECEIVE_T88 = 88,
    DRV_MXT336T_OBJECT_PEN_ACTIVESTYLUSTRANSMIT_T89 = 89,
    DRV_MXT336T_OBJECT_PEN_ACTIVESTYLUSWINDOW_T90 = 90,
    DRV_MXT336T_OBJECT_DEBUG_CUSTOMDATACONFIG_T91 = 91,
    DRV_MXT336T_OBJECT_PROCI_SYMBOLGESTUREPROCESSOR_T92 = 92,
    DRV_MXT336T_OBJECT_PROCI_TOUCHSEQUENCELOGGER_T93 = 93,
    DRV_MXT336T_OBJECT_SPT_PTCCONFIG_T95 = 95,
    DRV_MXT336T_OBJECT_SPT_PTCTUNINGPARAMS_T96 = 96,
    DRV_MXT336T_OBJECT_TOUCH_PTCKEYS_T97 = 97,
    DRV_MXT336T_OBJECT_PROCG_PTCNOISESUPPRESSION_T98 = 98,
    DRV_MXT336T_OBJECT_PROCI_KEYGESTUREPROCESSOR_T99 = 99,
    DRV_MXT336T_OBJECT_TOUCH_MULTITOUCHSCREEN_T100 = 100,
    DRV_MXT336T_OBJECT_SPT_TOUCHSCREENHOVER_T101 = 101,
    DRV_MXT336T_OBJECT_SPT_SELFCAPHOVERCTECONFIG_T102 = 102,
    DRV_MXT336T_OBJECT_PROCI_SCHNOISESUPPRESSION_T103 = 103,
    DRV_MXT336T_OBJECT_SPT_AUXTOUCHCONFIG_T104 = 104,
    DRV_MXT336T_OBJECT_SPT_DRIVENPLATEHOVERCONFIG_T105 = 105,
    DRV_MXT336T_OBJECT_SPT_ACTIVESTYLUSMMBCONFIG_T106 = 106,
    DRV_MXT336T_OBJECT_PROCI_ACTIVESTYLUS_T107 = 107,
    DRV_MXT336T_OBJECT_PROCG_NOISESUPSELFCAP_T108 = 108,
    DRV_MXT336T_OBJECT_SPT_SELFCAPGLOBALCONFIG_T109 = 109,
    DRV_MXT336T_OBJECT_SPT_SELFCAPTUNINGPARAMS_T110 = 110,
    DRV_MXT336T_OBJECT_SPT_SELFCAPCONFIG_T111 = 111,
    DRV_MXT336T_OBJECT_PROCI_SELFCAPGRIPSUPPRESSION_T112 = 112,
    DRV_MXT336T_OBJECT_SPT_PROXMEASURECONFIG_T113 = 113,
    DRV_MXT336T_OBJECT_SPT_ACTIVESTYLUSMEASCONFIG_T114 = 114,
    DRV_MXT336T_OBJECT_PROCI_SYMBOLGESTURE_T115 = 115,
    DRV_MXT336T_OBJECT_SPT_SYMBOLGESTURECONFIG_T116 = 116,
    DRV_MXT336T_OBJECT_GEN_INFOBLOCK16BIT_T254 = 254,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T220 = 220,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T221 = 221,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T222 = 222,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T223 = 223,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T224 = 224,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T225 = 225,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T226 = 226,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T227 = 227,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T228 = 228,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T229 = 229,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T230 = 230,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T231 = 231,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T232 = 232,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T233 = 233,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T234 = 234,
```

```
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T235 = 235,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T236 = 236,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T237 = 237,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T238 = 238,
    DRV_MXT336T_OBJECT_SPT_PROTOTYPE_T239 = 239,
    DRV_MXT336T_OBJECT_RESERVED_T255 = 255
} DRV_MXT336T_OBJECT_TYPE;
```

Description

Enumeration DRV_MXT336T_OBJECT_TYPE

The MAxtouch devices follow a Object protocol for their driver implementation. This makes it possible to implement a generic driver for many maxtouch devices. The device communicates the different properties or status like touch messages etc with the driver through an Object table. The different types of objects associated with the maxtouch device are listed in the enum below.

Remarks

None.

DRV_MXT_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_mxt.h

C

```
#define DRV_MXT_HANDLE_INVALID ((DRV_MXT_HANDLE)(-1))
```

Description

MXT Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_MXT_Open() and DRV_MXT_Close() functions if the request was not successful.

Remarks

None.

_DRV_MXT336T_H Macro

File

drv_mxt336t.h

C

```
#define _DRV_MXT336T_H
```

Description

This is macro _DRV_MXT336T_H.

DRV_MXT_I2C_MASTER_READ_ID Macro

MXT input read, I2C address from where master reads touch input data.

File

drv_mxt.h

C

```
#define DRV_MXT_I2C_MASTER_READ_ID 0x4B
```

Description

MXT Driver Module Master Input Read I2C address

This constant defines the MXT touch input read I2C address. This address is used as I2C address to read Touch input from MXT Touch controller.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific data sheets.

DRV_MXT_I2C_MASTER_WRITE_ID Macro

MXT command register write, I2C address where master sends the commands.

File

drv_mxt.h

C

```
#define DRV_MXT_I2C_MASTER_WRITE_ID 0x4A
```

Description

MXT Driver Module Master Command Write I2C Address

This constant defines the MXT command register I2C write address. This address is used as I2C address to write commands into MXT Touch controller register.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_MXT_I2C_READ_FRAME_SIZE Macro

I2C Frame size for reading MXT touch input.

File

drv_mxt.h

C

```
#define DRV_MXT_I2C_READ_FRAME_SIZE 7
```

Description

MXT Driver Module I2C Frame Size

This constant identifies the size of I2C frame required to read from MXT touch controller. MXT notifies the availability of input data through interrupt pin.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_MXT_INDEX_0 Macro

MXT driver index definitions.

File

drv_mxt.h

C

```
#define DRV_MXT_INDEX_0 0
```

Description

MXT Driver Module Index Numbers

These constants provide the MXT driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_MXT_Initialize and DRV_MXT_Open functions to identify the driver instance in use.

DRV_MXT_INDEX_1 Macro

File

drv_mxt.h

C

```
#define DRV_MXT_INDEX_1 1
```

Description

This is macro DRV_MXT_INDEX_1.

DRV_MXT_INDEX_COUNT Macro

Number of valid Touch controller MXT driver indices.

File

drv_mxt.h

C

```
#define DRV_MXT_INDEX_COUNT 2
```

Description

MXT Driver Module Index Count

This constant identifies the number of valid Touch Controller MXT driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_MXT336T_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_HANDLE_INVALID ((DRV_MXT336T_HANDLE)(-1))
```

Description

MXT336T Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_MXT336T_Open() and DRV_MXT336T_Close() functions if the request was not successful.

Remarks

None.

DRV_MXT336T_I2C_FRAME_SIZE Macro

I2C Frame size for reading MXT336T touch input.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_I2C_FRAME_SIZE 32
```

Description

MXT336T Driver Module I2C Frame Size

This constant identifies the size of I2C frame required to read from MXT336T touch controller. MXT336T notifies the availability of input data through interrupt pin.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_MXT336T_I2C_MASTER_READ_ID Macro

MXT336T input read, I2C address from where master reads touch input data.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_I2C_MASTER_READ_ID 0x95
```

Description

MXT336T Driver Module Master Input Read I2C address

This constant defines the MXT336T touch input read I2C address. This address is used as I2C address to read Touch input from MXT336T Touch controller.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific data sheets.

DRV_MXT336T_I2C_MASTER_WRITE_ID Macro

MXT336T command register write, I2C address where master sends the commands.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_I2C_MASTER_WRITE_ID 0x94
```

Description

MXT336T Driver Module Master Command Write I2C Address

This constant defines the MXT336T command register I2C write address. This address is used as I2C address to write commands into MXT336T Touch controller register.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_MXT336T_I2C_READ_ID_FRAME_SIZE Macro

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_I2C_READ_ID_FRAME_SIZE 8
```

Description

This is macro DRV_MXT336T_I2C_READ_ID_FRAME_SIZE.

DRV_MXT336T_INDEX_0 Macro

MXT336T driver index definitions.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_INDEX_0 0
```

Description

MXT336T Driver Module Index Numbers

These constants provide the MXT336T driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_MXT336T_Initialize and DRV_MXT336T_Open functions to identify the driver instance in use.

DRV_MXT336T_INDEX_1 Macro

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_INDEX_1 1
```

Description

This is macro DRV_MXT336T_INDEX_1.

DRV_MXT336T_INDEX_COUNT Macro

Number of valid Touch controller MXT336T driver indices.

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_INDEX_COUNT 2
```

Description

MXT336T Driver Module Index Count

This constant identifies the number of valid Touch Controller MXT336T driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific header files defined as part of the peripheral libraries.

t100_event Enumeration

Types of touch events reported by the Maxtouch Multi touch object

File

drv_mxt.h

C

```
enum t100_event {
  MXT_T100_EVENT_NO_EVENT = 0,
  MXT_T100_EVENT_MOVE = 1,
  MXT_T100_EVENT_UNSUP = 2,
  MXT_T100_EVENT_SUP = 3,
  MXT_T100_EVENT_DOWN = 4,
  MXT_T100_EVENT_UP = 5,
  MXT_T100_EVENT_UNSUPSUP = 6,
  MXT_T100_EVENT_UNSUPUP = 7,
  MXT_T100_EVENT_DOWNSUP = 8,
  MXT_T100_EVENT_DOWNUP = 9
};
```

Description

Enumeration: t100_event

The maxtouch multi touch object DRV_MXT336T_OBJECT_TOUCH_MULTITOUCHSCREEN_T100 return a number of different types of touch events. Each touch event has a return type associated with it. These are listed in this enum. These events are returned in the touch status message associated with the multi touch object.

Remarks

None.

t100_type Enumeration

Types of touch types reported by the Maxtouch Multi touch object

File

drv_mxt.h

C

```
enum t100_type {
  MXT_T100_TYPE_FINGER = 1,
  MXT_T100_TYPE_PASSIVE_STYLUS = 2,
  MXT_T100_TYPE_ACTIVE_STYLUS = 3,
  MXT_T100_TYPE_HOVERING_FINGER = 4,
  MXT_T100_TYPE_GLOVE = 5,
  MXT_T100_TYPE_LARGE_TOUCH = 6
};
```

Description

Enumeration: t100_type

The maxtouch multi touch object DRV_MXT336T_OBJECT_TOUCH_MULTITOUCHSCREEN_T100 return a number of different types of touch types. These are listed in this enum. The touch type is returned in the touch status message associated with the multi touch object.

Remarks

None.

DRV_MXT336T_T100_XRANGE Macro

MXT336T Driver Object Register Adresses for the registers being read in the driver

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_T100_XRANGE 13
```

Description

MXT336T Driver Object Register Adresses for the registers being read in the driver

MXT336T Objects have different registers that contain certain values regarding display resoltuion etc. These register addresses are used to read the values from object tables.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific protocol guides.

DRV_MXT336T_T100_YRANGE Macro

File

drv_mxt336t.h

C

```
#define DRV_MXT336T_T100_YRANGE 24
```

Description

This is macro DRV_MXT336T_T100_YRANGE.

Files

Files

Name	Description
drv_mxt.h	Touch controller MXT Driver interface header file.

drv_mxt336t.h	Touch controller MXT336T Driver interface header file.

Description

drv_mxt.h

Touch controller MXT Driver interface header file.

Enumerations

	Name	Description
	t100_event	Types of touch events reported by the Maxtouch Multi touch object
	t100_type	Types of touch types reported by the Maxtouch Multi touch object
	DRV_MXT_MODULE_ID	Number of valid MXT driver indices.
	DRV_MXT_TASK_STATE	Enumeration defining MXT touch controller driver task state.

Functions

	Name	Description
	DRV_MXT_Close	Closes an opened instance of the MXT driver. **Implementation:** Dynamic
	DRV_MXT_Deinitialize	Deinitializes the specified instance of the MXT driver module. **Implementation:** Dynamic
	DRV_MXT_Initialize	Initializes the MXT instance for the specified driver index. **Implementation:** Dynamic
	DRV_MXT_MaxtouchEventCallback	
	DRV_MXT_Open	Opens the specified MXT driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_MXT_Status	Provides the current status of the MXT driver module. **Implementation:** Dynamic
	DRV_MXT_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_MXT_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_MXT_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_MXT_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_MXT_TouchStatus	Returns the status of the current touch input.

Macros

	Name	Description
	DRV_MXT_HANDLE_INVALID	Definition of an invalid handle.
	DRV_MXT_I2C_MASTER_READ_ID	MXT input read, I2C address from where master reads touch input data.
	DRV_MXT_I2C_MASTER_WRITE_ID	MXT command register write, I2C address where master sends the commands.
	DRV_MXT_I2C_READ_FRAME_SIZE	I2C Frame size for reading MXT touch input.
	DRV_MXT_INDEX_0	MXT driver index definitions.
	DRV_MXT_INDEX_1	This is macro DRV_MXT_INDEX_1.
	DRV_MXT_INDEX_COUNT	Number of valid Touch controller MXT driver indices.

Structures

	Name	Description
	_DRV_MXT_CLIENT_OBJECT	MXT Driver client object maintaining client data.
	DRV_MXT_CLIENT_OBJECT	MXT Driver client object maintaining client data.
	DRV_MXT_INIT	Defines the data required to initialize or reinitialize the MXT driver
	DRV_MXT_OBJECT	Defines the data structure maintaining MXT driver instance object.
	DRV_MXT_TASK_QUEUE	Defines the MXT Touch Controller driver task data structure.

Types

	Name	Description
	DRV_MXT_HANDLE	Touch screen controller MXT driver handle.

Description

Touch Controller MXT Driver Interface File

This header file describes the macros, data structure and prototypes of the touch controller MXT driver interface.

File Name

drv_MXT.c

drv_mxt336t.h

Touch controller MXT336T Driver interface header file.

Enumerations

	Name	Description
	DRV_MXT336T_OBJECT_TYPE	The enum lists the different objects supported by the maxtouch device.

Functions

	Name	Description
	DRV_MXT336T_Close	Closes an opened instance of the MXT336T driver. **Implementation:** Dynamic
	DRV_MXT336T_CloseObject	Closes an opened instance of the MXT336T client object
	DRV_MXT336T_Deinitialize	Deinitializes the specified instance of the MXT336T driver module. **Implementation:** Dynamic
	DRV_MXT336T_DEVICE_ClientObjectEventHandlerSet	Sets the event handler for a MXT336T client object
	DRV_MXT336T_Initialize	Initializes the MXT336T instance for the specified driver index
	DRV_MXT336T_Open	Opens the specified MXT336T driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT336T_OpenObject	Opens the specified MXT336T object driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_MXT336T_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_MXT336T_Status	Provides the current status of the MXT336T driver module. **Implementation:** Dynamic
	DRV_MXT336T_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic

Macros

	Name	Description
	_DRV_MXT336T_H	This is macro _DRV_MXT336T_H.
	DRV_MXT336T_HANDLE_INVALID	Definition of an invalid handle.
	DRV_MXT336T_I2C_FRAME_SIZE	I2C Frame size for reading MXT336T touch input.
	DRV_MXT336T_I2C_MASTER_READ_ID	MXT336T input read, I2C address from where master reads touch input data.
	DRV_MXT336T_I2C_MASTER_WRITE_ID	MXT336T command register write, I2C address where master sends the commands.
	DRV_MXT336T_I2C_READ_ID_FRAME_SIZE	This is macro DRV_MXT336T_I2C_READ_ID_FRAME_SIZE.
	DRV_MXT336T_INDEX_0	MXT336T driver index definitions.
	DRV_MXT336T_INDEX_1	This is macro DRV_MXT336T_INDEX_1.
	DRV_MXT336T_INDEX_COUNT	Number of valid Touch controller MXT336T driver indices.
	DRV_MXT336T_T100_XRANGE	MXT336T Driver Object Register Adresses for the registers being read in the driver
	DRV_MXT336T_T100_YRANGE	This is macro DRV_MXT336T_T100_YRANGE.

Structures

	Name	Description
	DRV_MXT336T_OBJECT_CLIENT_EVENT_DATA	This structure maintains the information associated with each msg received or event that occurs

Types

	Name	Description
	DRV_MXT336T_CLIENT_CALLBACK	Pointer to a MXT336T client callback function data type.
	DRV_MXT336T_HANDLE	Touch screen controller MXT336T driver handle.
	DRV_MXT336T_INIT	Defines the data required to initialize or reinitialize the MXT336T driver

Description

Touch Controller MXT336T Driver Interface File

This header file describes the macros, data structure and prototypes of the touch controller MXT336T driver interface.

File Name

drv_MXT336T.c

USB Driver Libraries

Common Interface

Provides information on the USB Driver interface that is common to all PIC32 devices.

Description

The USB Driver Common Interface definition specifies the functions and their behavior that a USB Driver must implement so that the driver can be used by the MPLAB Harmony USB Host and Device Stack.

Note: The MPLAB Harmony USB Driver for PIC32MX and PIC32MZ devices implements the USB Driver Common Interface.

The USB Driver Common Interface contains functions that are grouped as follows:

- *Driver System Functions* - These functions are called by MPLAB Harmony to initialize and maintain the operational state of the USB Driver. The system functions can vary between different PIC32 device USB Drivers. As such, the USB Driver Common Interface does not require these functions to be of the same type. These functions are not called by the USB Host or Device Stack and therefore are allowed to (and can) vary across different PIC32 device USB Drivers. A description of these functions, along with a description of how to initialize the USB Driver for Host, Device or Dual Role operation, is provided in the specific PIC32 device USB Driver help section (see PIC32MX USB Driver and PIC32MZ USB Driver).
- *Driver General Client Functions* -These functions are called by the USB Host or Device Stack to gain access to the driver
- *Driver Host Mode Client Functions* - These functions are called exclusively by the USB Host Stack to operate and access the USB as a Host
- *Driver Device Mode Client Functions* - These functions are called exclusively by the USB Device Stack to operate and access the USB as a Device

The USB Driver Common Interface is defined in the `<install-dir>\framework\driver\usb\drv_usb.h` file. This file contains the data types and structures that define the interface. Specifically, the DRV_USB_HOST_INTERFACE structure, contained in this file, is the common interface for USB Driver Host mode functions. It is a structure of function pointers, pointing to functions that define the Driver Host mode Client functions. The following code example shows this structure and the function pointer it contains.

```
// *****************************************************************************
/* USB Driver Client Functions Interface (For Host mode)

  Summary:
    Group of function pointers to the USB Driver Host mode Client Functions.

  Description:
    This structure is a group of function pointers pointing to the USB Driver
    Host mode Client routines. The USB Driver should export this group of
    functions so that the Host layer can access the driver functionality.

  Remarks:
    None.
*/
```

```
typedef struct
{
    /* This is a pointer to the driver Open function. This function may be
     * called twice in a Dual Role application, once by the Host Stack and then
     * by the Device Stack */
    DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

    /* This is pointer to the driver Close function */
    void (*close)(DRV_HANDLE handle);

    /* This is a pointer to the event call back set function */
    void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData,
           DRV_USB_EVENT_CALLBACK eventHandler);

    /* This is a pointer to the Host IRP submit function */
    USB_ERROR (*hostIRPSubmit)(DRV_USB_HOST_PIPE_HANDLE pipeHandle, USB_HOST_IRP * irp);

    /* This is a pointer to the Host IRP Cancel all function */
    void (*hostIRPCancel)(USB_HOST_IRP * irp);

    /* This is pointer to the Host event disable function */
    bool (*hostEventsDisable)(DRV_HANDLE handle);

    /* This is a pointer to the Host event enable function */
    void (*hostEventsEnable)(DRV_HANDLE handle, bool eventContext);

    /* This is a pointer to the Host pipe setup function */
    DRV_USB_HOST_PIPE_HANDLE (*hostPipeSetup)
    (
        DRV_HANDLE client,
        uint8_t deviceAddress,
        USB_ENDPOINT endpointAndDirection,
        uint8_t hubAddress,
        uint8_t hubPort,
        USB_TRANSFER_TYPE pipeType,
        uint8_t bInterval,
        uint16_t wMaxPacketSize,
        USB_SPEED speed
    );

    /* This is a pointer to the Host Pipe Close function */
    void (*hostPipeClose)(DRV_USB_HOST_PIPE_HANDLE pipeHandle);

    /* This is a pointer to the Host Root Hub functions */
    DRV_USB_ROOT_HUB_INTERFACE rootHubInterface;

} DRV_USB_HOST_INTERFACE;
```

The DRV_USB_DEVICE_INTERFACE structure, contained in this file, is the common interface for USB Driver Device mode functions. It is a structure of function pointers, pointer to functions that define the Driver Device mode Client functions. The following code example shows this structure and the function pointer it contains.

```
// *****************************************************************************
/* USB Driver Client Functions Interface (For Device Mode)

  Summary:
    Group of function pointers to the USB Driver Device Mode Client Functions.

  Description:
    This structure is a group of function pointers pointing to the USB Driver
    Device Mode Client routines. The USB Driver should export this group of
    functions so that the Device Layer can access the driver functionality.

  Remarks:
    None.
*/

typedef struct
{
    /* This is a pointer to the driver Open function */
    DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

```
    /* This is pointer to the driver Close function */
    void (*close)(DRV_HANDLE handle);

    /* This is a pointer to the event call back set function */
    void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData,
           DRV_USB_EVENT_CALLBACK eventHandler);

    /* This is a pointer to the device address set function */
    void (*deviceAddressSet)(DRV_HANDLE handle, uint8_t address);

    /* This is a pointer to the device current speed get function */
    USB_SPEED (*deviceCurrentSpeedGet)(DRV_HANDLE handle);

    /* This is a pointer to the SOF Number get function */
    uint16_t (*deviceSOFNumberGet)(DRV_HANDLE handle);

    /* This is a pointer to the device attach function */
    void (*deviceAttach)(DRV_HANDLE handle);

    /* This is a pointer to the device detach function */
    void (*deviceDetach)(DRV_HANDLE handle);

    /* This is a pointer to the device endpoint enable function */
    USB_ERROR (*deviceEndpointEnable)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
                USB_TRANSFER_TYPE transferType, uint16_t endpointSize);

    /* This is a pointer to the device endpoint disable function */
    USB_ERROR (*deviceEndpointDisable)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device endpoint stall function */
    USB_ERROR (*deviceEndpointStall)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device endpoint stall clear function */
    USB_ERROR (*deviceEndpointStallClear)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is pointer to the device endpoint enable status query function */
    bool (*deviceEndpointIsEnabled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is pointer to the device endpoint stall status query function */
    bool (*deviceEndpointIsStalled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device IRP submit function */
    USB_ERROR (*deviceIRPSubmit)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
                USB_DEVICE_IRP * irp);

    /* This is a pointer to the device IRP Cancel all function */
    USB_ERROR (*deviceIRPCancelAll)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device remote wakeup start function */
    void (*deviceRemoteWakeupStart)(DRV_HANDLE handle);

    /* This is a pointer to the device remote wakeup stop function */
    void (*deviceRemoteWakeupStop)(DRV_HANDLE handle);

    /* This is a pointer to the device Test mode enter function */
    USB_ERROR (*deviceTestModeEnter)(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);

} DRV_USB_DEVICE_INTERFACE;
```

Both of these structures also contain pointers to General Client functions. The specific PIC32 device USB Driver allocates and initializes such a structure. The following code example shows how the PIC32MX USB Host mode Driver allocates and initializes the DRV_USB_HOST_INTERFACE structure. This code is contained in the `<install-dir>\framework\driver\usb\usbhs\src\dynamic\drv_usbfs_host.c` file.

```
/*****************************************************
 * This structure is a set of pointer to the USBFS driver
 * functions. It is provided to the Host layer as the
 * interface to the driver.
 * ****************************************************/
```

```
DRV_USB_HOST_INTERFACE gDrvUSBFSHostInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBFS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBFS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBFS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBFS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBFS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBFS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBFS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet =
                                                    DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
                                                    DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBFS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBFS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBFS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBFS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

Similarly, the PIC32MX USB Device mode Driver allocates and initializes the DRV_USB_DEVICE_INTERFACE structure. This can be reviewed in the <install-dir>\framework\driver\usb\usbhs\src\dynamic\drv_usbfs_device.c file.

```
/*************************************************
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * Device Layer when the Device Layer must use the
 * PIC32MX USB Controller.
 *************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBFSDeviceInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBFS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBFS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBFS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBFS_DEVICE_Attach,
    .deviceDetach = DRV_USBFS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBFS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBFS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBFS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBFS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBFS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBFS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBFS_DEVICE_IRPSubmit,
    .deviceIRPCancelAll = DRV_USBFS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBFS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBFS_DEVICE_RemoteWakeupStart,
    .deviceTestModeEnter = NULL


};
```

A pointer to the DRV_USB_HOST_INTERFACE structure is passed to the USB Host Stack as part of USB Host Stack initialization. The following code example shows how this is done.

```
/*********************************************************************************
 * This is a table of the USB Host mode drivers that this application will
 * support. Also contained in the driver index. In this example, the
 * application will want to use instance 0 of the PIC32MX USB Full-Speed driver.
 * ********************************************************************************/
const USB_HOST_HCD hcdTable =
{
    .drvIndex = DRV_USBFS_INDEX_0,
```

```
    .hcdInterface = DRV_USBFS_HOST_INTERFACE
};

/* Here the pointer to the USB Driver Common Interface is provided to the USB
 * Host Layer via the hostControllerDrivers member of the Host Layer
 * Initialization data structure. */
const USB_HOST_INIT usbHostInitData =
{
    .nTPLEntries = 1 ,
    .tplList = (USB_HOST_TPL_ENTRY *)USBTPList,
    .hostControllerDrivers = (USB_HOST_HCD *)&hcdTable

};
```

A pointer to the DRV_USB_DEVICE_INTERFACE structure is passed to the USB Device Stack as part of the USB Device Stack initialization. The Host Stack and Device Stack then access the driver functions through the function pointers contained in these structures.

The Driver General Client, Host mode and Device mode Client functions are described in this section. Any references to a USB Driver Client in the following sections, implies the client is a USB Host Stack and/or the USB Device Stack.

Driver General Client Functions

Provides information on the General Client functions for the USB Driver.

Description

The DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures contain pointers to the USB Driver's General Client functions. These functions are not specific to the operation mode (Host, Device, or Dual Role) of the driver. A USB Driver must implement these functions and ensure that the Host or Device Stack can access these functions through the driver's common interface structures. The common interface contains three general client functions:

- Driver Open Function
- Driver Close Function
- Driver Event Handler Set Function

Driver Open Function

The `open` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Open function. The signature of the Open function is as follows:

```
DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

The USB Driver Open function must match this signature. The Driver Client uses the USB Driver index (drvIndex) to specify the instance of the USB module that Host Stack or the Device Stack should open. The USB Driver should ignore the `intent` parameter. The function should return a driver handle. If the driver is not ready to be opened, it should return an invalid handle (DRV_HANDLE_INVALID). In such a case, the client will continue trying to open the driver by calling the Open function again. The driver may also fail to open for an invalid `index` parameter or if USB module is in an error condition.

When supporting Dual Role operation, both the Host Stack and Device Stack will call the Driver Open function in one application. The USB Driver must support multiple calls to the Open function in the same application. The Open function should be thread-safe.

Driver Close Function

The `close` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Close function. The signature of the Close function is as follows:

```
void (*close)(DRV_HANDLE handle);
```

The USB Driver Close function must match this signature. The Driver Client passes the handle obtained from the Driver Open function as a parameter to the close. The USB Host Stack or USB Device Stack will close the driver only when the stack is deinitialized (which is typically a rare case). The USB Driver should deallocate any client-related resources in the Close function. If the specified driver handle is not valid, the Close function should not have any side effects. The USB Driver expects the Close function to be called from the context of the thread in which the driver was opened; therefore, this function *is not* expected to be thread-safe.

Driver Event Handler Set Function

The `eventHandlerSet` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Event Handler Set function. The signature of the Event Handler Set function is as follows:

```
void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK eventHandler);
```

The USB Driver Event Handler Set function must match this signature. The signature of the Client Event Handling function should match DRV_USB_EVENT_CALLBACK. The USB Driver calls this function when it must communicate USB events to the client. The client can set the `eventHandler` parameter to NULL if it does not want to receive USB Driver events. The client will receive Host mode events if the USB Driver is operating in Host mode. It will receive Device mode events if the USB Driver is operating in Device mode. The DRV_USB_EVENT type enumeration contains all the possible events that the USB Driver would generate. The following code example shows the enumeration.

```
// *****************************************************************************
/* USB Driver Events Enumeration
```

```
  Summary:
    Identifies the different events that the USB Driver provides.

  Description:
    Identifies the different events that the USB Driver provides. The USB Driver
    should be able to provide these events.

  Remarks:
    None.
*/

typedef enum
{
    /* Bus error occurred and was reported. This event can be generated in both
     * Host and Device mode. */
    DRV_USB_EVENT_ERROR = 1,

    /* Host has issued a device Reset. This event occurs only in Device mode */
    DRV_USB_EVENT_RESET_DETECT,

    /* Resume detected while USB in suspend mode. This event can be generated in
     * both Host and Device mode. In Host mode, the events occurs when a remote
     * wakeup capable device has generated resume signaling. In Device mode,
     * this event will occur when the Host has issued resume signaling. */
    DRV_USB_EVENT_RESUME_DETECT,

    /* This event is generated in Device mode only. It occurs when the Host
     * suspends the bus and the bus goes idle. */
    DRV_USB_EVENT_IDLE_DETECT,

    /* This event is generated in Host mode and Device mode. In Host mode, this
     * event occurs when the device has stalled the Host. In Device mode, this
     * event occurs when the Host has accessed a stalled endpoint thus
     * triggering the device to send a STALL to the Host. */
    DRV_USB_EVENT_STALL,

    /* This event is generated in Host mode and Device mode. In Device mode,
     * this event occurs when a SOF has been generated by the Host. In Host
     * mode, this event occurs when controller is about to generate an SOF.
     * */
    DRV_USB_EVENT_SOF_DETECT,

    /* This event is generated in Device mode when the VBUS voltage is above
     * VBUS session valid. */
    DRV_USB_EVENT_DEVICE_SESSION_VALID,

    /* This event is generated in Device mode when the VBUS voltage falls
     * below VBUS session valid. */
    DRV_USB_EVENT_DEVICE_SESSION_INVALID,

} DRV_USB_EVENT;
```

This completes the discussion on the Driver General Client Functions.

Driver Host Mode Client Functions

Provides information on the Host mode Client functions for the USB Driver.

Description

The DRV_USB_HOST_INTERFACE structure contains pointers to the USB Driver's Host mode Client functions. These functions are only applicable when the USB module is operating as a USB Host. Along with the function pointers to the driver's Host mode specific functions, the DRV_USB_HOST_INTERFACE structure also contains another structure of function pointers of the type DRV_USB_ROOT_HUB_INTERFACE. This structure contains function pointers to the USB Driver's Root Hub functions. A USB Driver must implement these functions and ensure that the Host Stack can access these functions through the driver's DRV_USB_HOST_INTERFACE structure. The Driver Host mode Client functions in the DRV_USB_HOST_INTERFACE structure are:

- Driver Host Pipe Setup Function
- Driver Host Pipe Close Function

- Driver Host Events Disable Function
- Driver Host Events Enable Function
- Driver Host IRP Submit Function
- Driver Host IRP Cancel Function

Driver Host Pipe Setup Function

The `hostPipeSetup` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host Pipe Setup function. The signature of the Host Pipe Setup function is as follows:

```
DRV_USB_HOST_PIPE_HANDLE (*hostPipeSetup)  ( DRV_HANDLE client,  uint8_t deviceAddress,
                           USB_ENDPOINT endpointAndDirection, uint8_t hubAddress,
                           uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval,
                           uint16_t wMaxPacketSize, USB_SPEED speed);
```

The USB Driver Host mode Pipe Setup function must match this signature. The USB Host Stack calls this function to create a communication pipe to the attached device. The function parameters define the property of this communication pipe. The `driverHandle` parameter is the handle to the driver obtained through the driver Open function. The `deviceAddress` and the `endpointAddress` parameters specify the address of the USB device and the endpoint on this device to which this pipe must connect.

If the device is connected to the Host though a hub, `hubAddress` and `hubPort` must specify the address of the hub and port to which the device is connected. The USB Driver will use these parameters to schedule split transactions if the target device is a Low-Speed or Full-Speed device and is connected to the Host through a high-speed hub. If the device is connected directly to the Host, these parameters should be set to zero ('0').

The `pipeType` parameter specifies the type of USB transfers that this pipe would support. The `bInterval` parameter is interpreted as per the USB 2.0 Specification based on the transfer type and the speed of the pipe. The `wMaxPacketSize` parameter defines the maximum size of a transaction that the driver should use while transporting a transfer on the pipe. The Host layer will use the information obtained from the USB device descriptors of the attached device to decide the `wMaxPacketSize` parameter.

The Driver Host Pipe Setup function should be thread-safe, but does not have to be event safe. The Host layer (or the Host Client Drivers) will not, and should not attempt to create a pipe in an interrupt, and therefore, an event context. The function should return DRV_USB_PIPE_HANDLE_INVALID if the driver could not open the pipe. The driver may not be able to open a pipe due to incorrect function parameters or due to lack of resources.

Driver Host Pipe Close Function

The `hostPipeClose` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host Pipe Close function. The signature of the Host Pipe Close function is as follows:

```
void (*hostPipeClose)(DRV_USB_HOST_PIPE_HANDLE pipeHandle);
```

The USB Driver Host mode Pipe Close function must match this signature. The USB Host Stack calls this function to close communication pipes. The `pipeHandle` parameter is the pipe handle obtained from the Pipe Setup function. The Host Client Driver typically closes pipes when a device detach was detected. The Client Driver may also close pipes when a device configuration needs to change or when the Client Driver is being unloaded by the Host. The Pipe Close function has no side effect if the pipe handle is invalid. Closing the pipe will abort all I/O Request Packets (IRP) that are scheduled on the pipe. Any transaction in progress will complete. The IRP callback functions for each IRP scheduled in the pipe will be called with a USB_HOST_IRP_STATUS_ABORTED status.

The USB Driver Pipe Close function must be thread-safe and event-safe. The latter requirement allows the Pipe Close function to be called in the context of the device detach Interrupt Service Routine.

Driver Host Event Disable Function

The `hostEventsDisable` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host mode Driver Events Disable function. The signature of the Events Disable function is as follows:

```
bool (*hostEventsDisable)(DRV_HANDLE handle);
```

The USB Driver Host mode Driver Events Disable function must match this signature. The Host Stack will call this function when it wants to execute a section of code that should not be interrupted by the USB Driver. Calling this function should disable USB Driver event generation. The `handle` parameter is set to the driver handle obtained via the driver Open function. The function will return the present state of the event generation, whether it is enabled or disabled. The Host Stack will pass this value to the USB Driver Host mode Driver Events Enable function when it needs to enable the driver events.

Driver Host Events Enable Function

The `hostEventsEnable` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host mode Driver Events Enable function. The signature of the events enable function is as follows:

```
void (*hostEventsEnable)(DRV_HANDLE handle, bool eventContext);
```

The USB Driver Host mode Driver Events Enable function must match this signature. The USB Host Stack calls this function to re-enable the USB Driver Host mode Events (if they were enabled) after it called the USB Driver Host mode Events Disable function to disable driver events. The `handle` parameter is set to the driver handle obtained via the driver Open function. The `eventContext` parameter is set to the value returned by the Host mode Driver Events Disable function. The USB Driver will use the `eventContext` parameter to restore the event generation status (enabled or disabled) to what it was when the USB Driver Host mode Driver Events Disable function was called.

Driver Host IRP Submit Function

The `hostIRPSubmit` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host IRP Submit function. The

signature of the IRP Submit function is as follows:
```
USB_ERROR (*hostIRPSubmit)(DRV_USB_HOST_PIPE_HANDLE pipeHandle, USB_HOST_IRP * irp);
```

The USB Driver Host IRP Submit function must match this signature. The Host Stack calls this function to submit an IRP to the USB Driver. The USB Driver provides this mechanism to transfer data between the Host Stack and the attached device. The `pipeHandle` parameter should be set to the pipe handle obtained by the Pipe Setup function. The pipe handle specifies the pipe, and therefore, the target device, endpoint, speed and transfer type, on which the I/O must be processed. The `irp` parameter should point to the IRP data structure. The IRP data structure will transport an entire transfer over the pipe. The USB Driver will split up the transfer into transactions based on the parameters specified at the time of pipe creation. This process does not require Host Stack intervention.

The function will return USB_ERROR_HOST_PIPE_INVALID if the pipe handle is not valid. It will return USB_ERROR_OSAL_FUNCTION if an error occurred while performing a RTOS-related operation. It will return USB_ERROR_NONE if the IRP was submitted successfully.

The USB Driver will queue the IRP if there is already an IRP being processed on the pipe. The completion of the IRP processing is indicated by the USB Driver calling the IRP Callback function specified within the IRP. The Host IRP Submit function must be thread-safe and IRP callback-safe. The Host Stack may resubmit the IRP within the IRP Callback function. The IRP Callback function itself executes within an interrupt context. The completion status of the IRP will be available in the `status` member of the IRP when the IRP callback function is invoked.

Driver Host IRP Cancel Function

The `hostIRPCancel` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host IRP Cancel function. The signature of the IRP Cancel function is as follows
```
void (*hostIRPCancel)(USB_HOST_IRP * irp);
```

The USB Driver Host IRP Cancel function must match this signature. The Host Stack and Host Client Drivers will call this function to cancel an IRP that was submitted. The IRP will be aborted successfully if it is not in progress. If the IRP processing has begun, the on-going transaction will complete and pending transactions in the transfer will be aborted. In either case, the IRP Callback function will be called with the IRP status as USB_HOST_IRP_STATUS_ABORTED.

Driver Host USB Root Hub Port Interface

Provides information on the Root Hub Port interface of the USB Host Driver.

Description

The `rootHubPortInterface` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Port functions. The data type of this member is USB_HUB_INTERFACE. This data type is a structure containing function pointers pointing to the port control functions of the root hub. The USB Driver must assign the function pointers in this structure to the root hub port control functions. These same functions are also exported by a Hub Driver to the USB Host Stack, which allow the Host Stack to control a device regardless of whether it is connected to a root hub or an external hub. The port functions are valid only when a device is attached to the port. The behavior of these functions on a port to which no device is connected is not defined. Descriptions of the port control functions are provided, which include:

- Driver Host Hub Port Reset Function
- Driver Host Hub Port Reset Completion Status Function
- Driver Host Hub Port Suspend Function
- Driver Host Hub Port Resume Function
- Driver Host Hub Port Speed Get Function

Driver Host Hub Port Reset Function

The `hubPortReset` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Reset function. The signature of this function is as follows:
```
USB_ERROR (*hubPortReset)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Reset function must follow this signature. This function starts reset signaling on the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. The USB Host Stack uses the parent identifier provided by the root hub driver when the USB_HOST_DeviceEnumerate function was called to query the driver handle that is linked to this root hub. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the function was successful. If the reset signaling is already in progress on the port, calling this function has no effect. The USB Driver will itself time duration of the reset signal. This does not require USB Host Stack intervention. The USB Host Stack will call the port reset completion status function to check if the reset signaling has completed. Calling this function on a port which exists on an external hub will cause the hub driver to issue a control transfer to start the port reset procedure.

Driver Host Hub Port Reset Completion Status Function

The `hubPortResetIsComplete` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Reset Completion Status function. The signature of this function is as follows:
```
bool (*hubPortResetIsComplete)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Reset Completion Status function must follow this signature. The USB Host Stack calls this function to check if the port reset sequence that was started on a port has completed. The function returns true if the reset signaling has completed. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function.

If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected.

Driver Host Hub Port Suspend Function

The `hubPortSuspend` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Suspend function. The signature of this function is as follows:

```
USB_ERROR(*hubPortSuspend)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Suspend function must follow this signature. The USB Host Stack calls this function to suspend the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the request was successful. Calling this function on a suspended port will not have any effect.

Driver Host Hub Port Resume Function

The `hubPortResume` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Resume function. The signature of this function is as follows:

```
USB_ERROR(*hubPortResume)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Resume function must follow this signature. The USB Host Stack calls this function to resume a suspended port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the request was successful. Calling this function on a port that is not suspended will not have any effect.

Driver Host Hub Port Speed Get Function

The `hubPortSpeedGet` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Speed Get function. The signature of this function is as follows:

```
USB_SPEED(*hubPortSpeedGet)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Speed Get function must follow this signature. The USB Host Stack calls this function to obtain the USB speed of the device that is attached to the port. The Host Stack calls this function only after it has completed reset of the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_SPEED_ERROR if the request was not successful. It will return the functional USB speed otherwise.

This concludes the section describing the USB Driver Host mode Client Functions. The USB Driver Device Mode Client Functions are discussed in the next section.

Driver Host Root Hub Interface

Provides information on the Root Hub interface for the USB Host Driver.

Description

The USB Driver Common Interface requires the USB Driver to be operating in Host mode to provide root hub control functions. If the USB peripheral does not contain root hub features in hardware, these features must be emulated in software by the driver. The USB peripheral on PIC32MX and PIC32MZ devices does not contain root hub features; therefore, the USB Driver for these peripherals emulates the root hub functionality in software. The `rootHubInterface` member of the DRV_USB_HOST_INTERFACE structure is a structure of type DRV_USB_ROOT_HUB_INTERFACE. The members of this structure are function pointers to the root hub control functions of the USB Driver.

Along with other Host mode functions, the USB Driver while operating in Host mode must also ensure that the `rootHubInterface` member of DRV_USB_HOST_INTERFACE is set up correctly so that the USB Host Stack can access the root hub functions. Descriptions of the function pointer types in the DRV_USB_ROOT_HUB_INTERFACE include:

- Driver Host Root Hub Speed Get Function
- Driver Host Root Hub Port Numbers Get Function
- Driver Host Root Hub Maximum Current Get Function
- Driver Host Root Hub Operation Enable Function
- Driver Host Root Hub Operation Enable Status Function
- Driver Host Root Hub Initialize Function

Driver Host Root Hub Speed Get Function

The `rootHubSpeedGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Speed Get

function. The signature of this function is as follows:
```
USB_SPEED (*rootHubSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Speed Get function must match this signature. The USB Host Stack calls this function to identify the speed at which the root hub is operating. The `handle` parameter is the handle obtained by calling the USB Driver Open function. The operation speed is configured by the USB Driver initialization and depends on the capability of the USB peripheral. For example, the USB peripheral on PIC32MZ devices supports both Hi-Speed and Full-Speed Host mode operation. It can be configured through initialization to only operate at Full-Speed. The Root Hub Speed Get function must return the USB speed at which the USB peripheral is operating. This should not be confused with the speed of the attached device.

Driver Host Root Hub Port Numbers Get Function

The `rootHubPortNumbersGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Numbers Get function. The signature of this function is as follows:
```
USB_SPEED (*rootHubSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Speed Get function must match this signature. This function should return the number of ports that the root hub contains. On the USB peripheral for both PIC32MZ and PIC32MX devices, this value is always '1'.

Driver Host Root Hub Maximum Current Get Function

The `rootHubMaxCurrentGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Maximum Current Get function. The signature of this function is as follows:
```
uint32_t (*rootHubMaxCurrentGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Maximum Current Get function must match this signature. This function returns the maximum VBUS current that the root hub can provide. The USB Host Stack calls this function to know the maximum current that the root hub VBUS power supply can provide. This value is then used to determine if the Host can support the current requirements of the attached device. The `handle` parameter is the driver handle obtained by calling the driver Open function.

The PIC32MX and the PIC32MZ USB peripherals cannot supply VBUS. The root hub driver only switches the VBUS supply. The current rating of the VBUS is specified through the USB Driver initialization. The root hub maximum current get function implementation in these drivers returns this value to the Host Stack.

Driver Host Root Hub Operation Enable Function

The `rootHubOperationEnable` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Operation Enable function. The signature of this function is as follows"
```
void (*rootHubOperationEnable)(DRV_HANDLE handle, bool enable);
```

The USB Driver Root Hub Operation Enable function must match this signature. The USB Host Stack calls this function when it ready to receive device attach events from the root hub. Calling this function will cause the USB Driver root hub functionality to enable detection of device attach and detach. The USB Driver will then raise events to the USB Host Stack. The `handle` parameter is the driver handle obtained by calling the driver Open function. Setting the `enable` parameter to true enables the root hub operation. Setting the `enable` parameter to false disables the root hub operation.

Driver Host Root Hub Operation Enable Status Function

The `rootHubOperationIsEnabled` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Operation Enable Status function. The signature of this function is as follows:
```
bool (*rootHubOperationIsEnabled)(DRV_HANDLE handle);
```

The USB Driver Root Hub Operation Enable Status function must match this signature. This USB Host Stack calls this function after calling the operation enable function to check if this has completed. The function returns true if the operation enable function has completed. The USB Host Stack will call this function periodically until it returns true.

Driver Host Root Hub Initialize Function

The `rootHubInitialize` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Initialize function. The signature of this function is as follows:
```
void (*rootHubInitialize)(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

The USB Driver Root Hub Initialize function must match this signature. The USB Host Stack calls this function to assign a device identifier (usbHostDeviceInfo) to the root hub. This function is called before the Host Stack enables the root hub operation. The USB Driver root hub should use this identifier as the parent identifier when it calls the USB_HOST_DeviceEnumerate function to enumerate the attached device. At the time of enumeration, the USB Host Stack will use this parent identifier to identify the parent hub (whether root hub or external hub) of the attached device. The USB Driver root hub should retain the `usbHostDeviceInfo` parameter for the life time of its operation.

Driver Device Mode Client Functions

Provides information on the USB Driver Device mode Client functions.

Description

The DRV_USB_DEVICE_INTERFACE structure contains pointers to the USB Driver's Device mode Client Functions. These functions are only applicable when the USB module is operating as a USB Device. A USB Driver must implement these functions and ensure that the Device Stack

can access these functions through the driver's DRV_USB_DEVICE_INTERFACE structure. Descriptions of the Driver Device Mode Client functions in the DRV_USB_DEVICE_INTERFACE structure include:

- Driver Device Address Set Function
- Driver Device Current Speed Get Function
- Driver Device SOF Number Get Function
- Driver Device Attach Function
- Driver Device Detach Function
- Driver Device Endpoint Enable Function
- Driver Device Endpoint Disable Function
- Driver Device Endpoint Stall Function
- Driver Device Endpoint Stall Clear Function
- Driver Device Endpoint Enable Status Function
- Driver Device Endpoint Stall Status Function
- Driver Device IRP Submit Function
- Driver Device IRP Cancel All Function
- Driver Device IRP Cancel Function
- Driver Device Remote Wakeup Start Function
- Driver Device Remote Wakeup Stop Function
- Driver Device Test Mode Enter Function

The PIC32MZ and the PIC32MX USB peripheral drivers implement the Device mode functions and export these functions to the Device Stack though their respective DRV_USB_DEVICE_INTERFACE structure.

Driver Device Address Set Function

The `deviceAddressSet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Address Set function. The signature of this function is as follows:

```
void (*deviceAddressSet)(DRV_HANDLE handle, uint8_t address);
```

The USB Driver Device Address Set Function should match this signature. The USB Device Stack will call this function to set the Device USB Address. The function will be called in an interrupt context and hence the function implementation must be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `address` parameter is the address provided by the USB Host through the Set Device Address Standard request.

Driver Device Current Speed Get Function

The `deviceCurrentSpeedGet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Current Speed Get function. The signature of this function is as follows:

```
USB_SPEED (*deviceCurrentSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Device Current Speed Get function should match this signature. The USB Device Stack will call this function to obtain the speed at which the device has connected to the USB. It will call this function after reset signaling has completed. The `handle` parameter is driver handle obtained from calling the driver Open function. This function is called in an interrupt context and should be interrupt-safe.

Driver Device SOF Number Get Function

The `deviceSOFNumberGet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Start-Of-Frame Number Get function. The signature of this function is as follows:

```
uint16_t (*deviceSOFNumberGet)(DRV_HANDLE handle);
```

The USB Driver SOF Number Get function should match this signature. The USB Device Stack will call this function to obtain the current SOF number. The USB peripheral uses a 16 bit counter to count the number of SOFs that have occurred since USB reset. This value is returned along with the Device Stack Start of Frame event. This function is called from an interrupt context and should be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Attach Function

The `deviceAttach` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Attach function. The signature of this function is as follows:

```
uint16_t(*deviceAttach)(DRV_HANDLE handle);
```

The USB Driver Attach function should match this signature. The USB Device Stack will call this function when the Device application calls the USB Device Stack Device Attach function. The USB Driver will enable the required signaling resistors for indicate attach to the Host. The application could call this function in response to a VBUS power available event. This function must be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Detach Function

The deviceDetach member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Detach function. The signature of this function is as follows:

uint16_t(*deviceDetach)(DRV_HANDLE handle);

The USB Driver Detach function should match this signature. The USB Device Stack will call this function when the Device application calls the USB Device Stack Device Detach function. The USB Driver will disable the required signaling resistors to indicate detach to the Host. The application could call this function in response to a VBUS power not available event. This function should be interrupt-safe. The `handle` parameter is driver handle obtained from calling the driver Open function.

Driver Device Endpoint Enable Function

The `deviceEndpointEnable` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Enable function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointEnable)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
            USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

The USB Driver Endpoint Enable function should match this signature. The USB Device Stack Function Driver will call this function when it is initialized by the USB Device Layer. The Device Layer, on receiving the Set Configuration request from the Host, identifies the function drivers that are required by the configuration and initializes them. The function drivers will call the endpoint enable function to enable the endpoints required for their operation. Enabling the endpoint will cause it reply to transaction requests from the Host and accept transfer requests from the device application.

The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be enabled. The transferType is the type of the USB transfer that this endpoint will handle. The endpointSize is the size of the maximum transaction that the endpoint will handle. This should match the endpoint size communicated to the Host via the device endpoint descriptors.

The function will return USB_ERRROR_NONE if the endpoint was configured successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

The endpoint enable function will be called in an interrupt context and should be interrupt-safe. It is not expected to be thread safe. For standard function drivers, the endpoint enable function will be called in the context of the USB Device Layer Client. For vendor USB devices, the vendor application must call the endpoint enable function in response to and within the context of the device configured event. Again this event itself will execute in the context of the Device Layer.

Driver Device Endpoint Disable Function

The `deviceEndpointDisable` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Disable function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointDisable)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Disable function should match this signature. The USB Device Stack Function Driver will call this function when it is deinitialized by the USB Device Layer. The Device Layer will deinitialize function drivers when it receives a USB reset event from the driver or on receiving the Set Configuration request from the Host with configuration parameter 0. Disabling the endpoint will cause it NAK transaction request from the Host and not accept transfer requests from the device application.

The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be disabled.

The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

The endpoint disable function will be called in an interrupt context and should be interrupt-safe. It is not expected to be thread safe. For standard function drivers, the endpoint disable function will be called in the context of the USB Device Layer Client. For vendor USB devices, the vendor application must call the endpoint enable function in response to and within the context of the device reset event. Again this event itself will execute in the context of the Device Layer. Disabling the endpoint will not cancel any transfers that have been queued against the endpoint. The function drivers will call the IRP Cancel All function to cancel any pending transfers.

Driver Device Endpoint Stall Function

The `deviceEndpointStall` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointStall)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall function should match this signature. The USB Device Stack Function Driver will call this function to stall an endpoint. The Device Layer itself will stall endpoint 0 for several reasons including non-support of the Host request or failure while executing the request. A function driver will also stall an endpoint for protocol specific reasons. The driver will stall both, receive and transmit directions when stalling Endpoint 0. The driver will stall the specified direction while stalling a non-zero endpoint.

This function must be thread safe and interrupt safe. Stalling the endpoint will abort all the transfers queued on the endpoint with the completion status set to USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be stalled. The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device Endpoint Stall Clear Function

The `deviceEndpointStallClear` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall Clear function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointStallClear)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall Clear function should match this signature. The USB Device Stack Function Driver will call this function to clear the stall on a non-zero endpoint. The Device Layer will call this function to clear the stall condition on Endpoint 0. Clearing the stall on a non-zero endpoint will clear all transfers scheduled on the endpoint and transfer completion status will be set to USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST. When the stall is cleared, the data toggle for non-zero endpoint will be set to DATA0. The data toggle on Endpoint 0 OUT endpoint will be set to DATA1. The USB Driver will clear the Stall condition on an endpoint even if it was not stalled.

This function must be thread safe and interrupt safe. Stalling the endpoint will flush all the transfers queued on the endpoint. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose stall condition must be cleared. The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device Endpoint Enable Status Function

The `deviceEndpointIsEnabled` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Enable Status function. The signature of this function is as follows:

```
bool (*deviceEndpointIsEnabled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Enable Status function should match this signature. The USB Device Stack function will call this function to check if an endpoint has been enabled. The function returns true if the endpoint is enabled. The endpoint is enabled through the USB Driver Endpoint Enable function. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose enable status needs to be queried.

Driver Device Endpoint Stall Status Function

The `deviceEndpointIsStalled` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall Status function. The signature of this function is as follows:

```
bool (*deviceEndpointIsStalled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall Status function should match this signature. The USB Device Stack function will call this function to check if an endpoint has been stalled. The function returns true if the endpoint is stalled. The endpoint is stalled through the USB Driver Endpoint Stall function. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose stall status needs to be queried.

Driver Device IRP Submit Function

The deviceIRPSubmit member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Submit function. The signature of the IRP submit function is as follows:

```
USB_ERROR (*deviceIRPSubmit)(DRV_HANDLE handle, USB_ENDPOINT endpoint, USB_DEVICE_IRP * irp);
```

The USB Driver Device IRP Submit function must match this signature. The Device Stack (USB Device calls this function to submit an IRP to the USB Driver. The USB Driver provides this mechanism to transfer data between the device and the Host. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter should set to endpoint through which transfer must be processed. The `irp` parameter should point to the Device IRP data structure. The IRP data structure will transport an entire transfer over the endpoint. The USB Driver will split up the transfer into transactions based on the endpoint size specified at the time of enabling the endpoint. This process does not require Device Stack intervention.

The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid. It will return USB_ERROR_DEVICE_IRP_IN_USE if an in progress IRP is resubmitted. It will return USB_ERROR_ENDPOINT_NOT_CONFIGURED if the IRP is submitted to an endpoint that is not enabled.

The USB Driver will queue the IRP if there is already an IRP being processed on the endpoint. The completion of the IRP processing is indicated by the USB Driver calling the IRP callback function specified within the IRP. The Device IRP Submit function must be thread safe and IRP callback safe. The Device Stack may resubmit the IRP within the IRP callback function. The IRP callback function itself executes within an interrupt context. The completion status of the IRP will be available in the status member of the IRP when the IRP callback function is invoked.

Driver Device IRP Cancel All Function

The `deviceIRPCancelAll` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Cancel All function. The signature of this is as follows:

```
USB_ERROR (*deviceIRPCancelAll)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Device IRP Cancel All function must match this signature. The USB Device Stack will call this function before disabling the endpoint. Calling this function will call all IRPs that are queued on the endpoint to be canceled. The callback of each IRP will be invoked and the IRP completion status will be set to USB_DEVICE_IRP_STATUS_ABORTED. If an IRP is in progress, an ongoing transaction will be allowed to complete and pending transactions will be canceled. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose queued IRPs must be canceled.

The function is thread safe and interrupt safe and will return USB_ERRROR_NONE if it executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device IRP Cancel Function

The `deviceIRPCancel` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Cancel function. The signature of this is as follows:

```
USB_ERROR (*deviceIRPCancel)(DRV_HANDLE handle, USB_DEVICE_IRP * IRP);
```

The USB Driver Device IRP Cancel function must match this signature. This function is called by the USB Device Stack function driver to cancel a scheduled IRP. If the IRP is in the queue but it's processing has not started, the IRP will removed from the queue and the IRP callback function will be called from within the cancel function. The callback will be invoked with the IRP completion status set to USB_DEVICE_IRP_STATUS_ABORTED. If an IRP is in progress, an ongoing transaction will be allowed to complete and pending transactions will be canceled. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `irp` parameter is the IRP to be canceled.

The function is thread safe and will return USB_ERRROR_NONE if it executed successfully. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid or if the IRP has status indicates that this IRP is not queued or not in progress. The application should not release the data memory associated with IRP unless the callback has been received.

Driver Device Remote Wakeup Start Function

The `deviceRemoteWakeupStart` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Remote Wakeup Start function. The signature of this function is as follows:

```
void (*deviceRemoteWakeupStart)(DRV_HANDLE handle);
```

The USB Driver Device Remote Wakeup Start function must match this signature. The USB Device Stack will call the function when the device application wants to start remote wakeup signaling. This would happen if the device supports remote wake-up capability and this has been enabled by the Host. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Remote Wakeup Stop Function

The `deviceRemoteWakeupStop` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Remote Wakeup Stop function. The signature of this function is as follows:

```
void (*deviceRemoteWakeupStop)(DRV_HANDLE handle);
```

The USB Driver Device Remote Wakeup Stop function must match this signature. The USB Device Stack will call the function when the device application wants to stop remote wakeup signaling. The application would call after calling the remote wakeup start function. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Test Mode Enter Function

The `deviceTestModeEnter` parameter of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Test Mode Enter function. The signature of this function is as follows:

```
USB_ERROR (*deviceTestModeEnter)(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

The USB Driver Device Test Mode Enter function should match this signature. The USB Device Stack calls this driver function to place the driver into test mode. This is required when the USB Host (operating at Hi-Speed) send the Set Feature request with the feature selector test set to test mode. This request also specifies which of the test mode signals, the driver should enable. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `testMode` parameter should be set to one of the test modes as defined in table 9-7 of the USB 2.0 specification.

The test mode enter function is only supported by the PIC32MZ USB Driver as the USB peripheral on this controller supports Hi-Speed operation. The function will return USB_ERRROR_NONE if it executed successfully. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

This concludes the discussion on the DRV_USB_DEVICE_INTERFACE structure. The following sections describe using the USB Common Driver.

Opening the Driver

Provides information and examples for opening the driver.

Description

The USB Host Stack and the USB Device Stack must obtain a handle to the USB Driver to access the functionality of the driver. This handle is obtained through the USB Driver Open function. The DRV_USB_DEVICE_INTERFACE structure and DRV_USB_DEVICE_HOST_INTERFACE structure provide access to the USB Driver Open function through the `open` member of these structures. Calling the Open function may not return a valid driver handle the first time the function is called. In fact, the USB Driver will return an invalid driver handle until the driver is ready to be opened. The Host and the Device Stack call the Open function repetitively in a state machine, until the function returns a valid handle.

The USB Host Stack can open the USB Driver but can call its Host mode functions only if the USB Driver was initialized for Host mode or Dual Role operation. The USB Host Stack accesses the driver functions through the DRV_USB_HOST_INTERFACE pointer that was provided to the Host Layer through the Host Stack initialization. The USB Device Stack can open the USB Driver but can call its Device mode functions only if the USB Driver was initialized for Device mode or Dual Role operation. The USB Device Stack accesses the driver functions through the DRV_USB_HOST_INTERFACE pointer that was provided to the Host Layer through the Host Stack initialization

The following code example shows how the USB Host Layer opens the USB Driver.

```
/* This code example shows how the Host Layer open the HCD via the hcdInterface.
 * The driver handle is stored in hcdHandle member of the busObj data structure.
```

```
 * The busObj data structure Host Layer local data structure. The Host Layer
 * opens the HCD when the bus is enabled. This operation takes place in the
 * USB_HOST_BUS_STATE_ENABLING state. */

/* Note the Host Layer calls the Open function by accessing the open member of
 * the hcdInterface which is of the type DRV_USB_HOST_INTERFACE. Also note how
 * the function is called repetitively until the Open function returns a valid
 * handle. */

case USB_HOST_BUS_STATE_ENABLING:

    /* The bus is being enabled. Try opening the HCD */
    busObj->hcdHandle = busObj->hcdInterface->open(busObj->hcdIndex, DRV_IO_INTENT_EXCLUSIVE |
            DRV_IO_INTENT_NONBLOCKING | DRV_IO_INTENT_READWRITE );

    /* Validate the Open function status */
    if (DRV_HANDLE_INVALID == busObj->hcdHandle )
    {
        /* The driver may not open the first time. This is okay. We
         * should try opening it again. The state of bus is not
         * changed. */
    }
```

The following code example shows how the USB Device Layer opens the USB Driver.

```
/* This code example shows how the USB Device Layer calls the USBCD open
 * function to open the USBCD. The Device Layer accesses the USBCD Open function
 * through the driverInterface member of the usbDeviceInstanceState object. The
 * driverInterface member is a DRV_USB_DEVICE_INTERFACE type. The
 * usbDeviceInstanceState is a USB Device Layer local object. */

/* The Device Layer attempts to open the USBCD when it is initializing. Note how
 * the Device Layer advances to the next state only when the USBCD returns a
 * valid handle.   */

switch(usbDeviceThisInstance->taskState)
{
    case USB_DEVICE_TASK_STATE_OPENING_USBCD:

        /* Try to open the driver handle. This could fail if the driver is
         * not ready to be opened. */
        usbDeviceThisInstance->usbCDHandle =
           usbDeviceThisInstance->driverInterface->open( usbDeviceThisInstance->driverIndex,
           DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

        /* Check if the driver was opened */
        if(usbDeviceThisInstance->usbCDHandle != DRV_HANDLE_INVALID)
        {
            /* Yes the driver could be opened. */

            /* Advance the state to the next state */
            usbDeviceThisInstance->taskState = USB_DEVICE_TASK_STATE_RUNNING;

            /* Update the USB Device Layer state to indicate that it can be
             * opened */
            usbDeviceThisInstance->usbDeviceInstanceState = SYS_STATUS_READY;
        }

        break;
```

USB Driver Host Mode Operation

Provides information on Host mode operation.

Description

The USB Driver operates or can operate in the Host mode when it is initialized for Host mode or Dual Role operation. When operating in Host mode, the USB Driver is also referred to as the Host Controller Driver (HCD). In Dual Role mode, the USB Driver will switch to Host mode when the USB Driver Host Root Hub Operation Enable function is called.

The USB Driver Client must perform these steps to operate the USB Driver in Host mode.

1. Open the USB Driver to obtain the driver handle.
2. Set the event handler.
3. Call the Root Hub Control function to obtain the speed of the root hub, the number of ports that the root hub supports, and the maximum current that the root hub VBUS can supply.
4. Calls the Root Hub Initialize function with an identifier parameter. This `identifier` parameter allows the Host Stack to uniquely identify the root hub when where there are multiple root hubs.
5. The Driver Client will then enable the root hub operation and will wait until the root hub operation is enabled.
6. The Driver Client can now call the USB Driver Host mode functions.

The following sections explain Steps 2 through 6 in more detail.

Handling Host Mode Driver Events

Currently, the HCD does not provide any events to the client. The client can optionally register an event handler through the eventHandlerSet function pointer in the DRV_USB_HOST_INTERFACE structure. Future releases of the USB Driver may contain features that provide events to the Driver Client. Please refer to the following **Root Hub Operation** section for details on how the driver indicates device attach and detach to the client.

Root Hub Operation

A key feature of the HCD is the Root Hub Driver. The Root Hub Driver emulates hub operation in USB Driver software and provides a hub like interface to the USB Host Layer. The USB Host Layer treats the root hub like an external hub. This simplifies the implementation of USB Host Layer while supporting multiple devices through a hub. In that, the USB Host layer does not have to treat a device connected directly to the USB peripheral differently than a device connected to an external hub. The following code example shows how the USB Host Layer calls the root hub function to obtain information about the root hub.

```
/* This code example shows how the USB Host Layer calls the root hub functions to
 * obtain information about the root. The USB Host Layer first opens the HCD and
 * then accesses the root hub functions through the rootHubInterface member of
 * hcdInterface. rootHubInterface is of the type DRV_USB_ROOT_HUB_INTERFACE and
 * the hcdInterface is of the type of DRV_USB_HOST_INTERFACE. */

/* The code example shows how the Host Layer gets to know the root hub operation
 * speed, number of root hub ports and the maximum amount of current that the
 * root can supply. These function can be called only after HCD was opened and a
 * valid driver handle obtained. */

case USB_HOST_BUS_STATE_ENABLING:

    /* The bus is being enabled. Try opening the HCD */
    busObj->hcdHandle = busObj->hcdInterface->open(busObj->hcdIndex, DRV_IO_INTENT_EXCLUSIVE |
            DRV_IO_INTENT_NONBLOCKING | DRV_IO_INTENT_READWRITE );

    /* Validate the Open function status */
    if (DRV_HANDLE_INVALID == busObj->hcdHandle )
    {
        /* The driver may not open the first time. This is okay. We
         * should try opening it again. The state of bus is not
         * changed. */
    }
    else
    {
        /* Update the bus root hub information with the
         * details of the controller. Get the bus speed, number of
         * ports, the maximum current that the HCD can supply,
         * pointer to the root hub port functions. */

        SYS_DEBUG_PRINT(SYS_ERROR_INFO,
             "\r\nUSB Host Layer: Bus %d Root Hub Driver Opened.",hcCount);

        busObj->rootHubInfo.speed =
             busObj->hcdInterface->rootHubInterface.rootHubSpeedGet(busObj->hcdHandle);

        busObj->rootHubInfo.ports =
             busObj->hcdInterface->rootHubInterface.rootHubPortNumbersGet(busObj->hcdHandle);

        busObj->rootHubInfo.power =
             busObj->hcdInterface->rootHubInterface.rootHubMaxCurrentGet(busObj->hcdHandle);

        busObj->rootHubInfo.rootHubPortInterface =
             busObj->hcdInterface->rootHubInterface.rootHubPortInterface;
```

The USB Host Layer must initialize and enable the operation of the root hub. While initializing the Root Hub Driver, the Host layer will assign a unique identifier to the root hub. The root hub will return this value as the parent identifier while calling the USB_HOST_DeviceEnumerate function. The USB Host Layer must then enable the operation of the root hub driver. This will cause the root hub driver to detect device attach and detach. The following code example shows how the USB Host Layer initializes and enables the root hub driver

```
/* The following code example show how the USB Host Layer initializes the root
 * hub and then enables the root hub operation. The
 * rootHubDevice->deviceIdentifier is a unique identifier that allows the USB
 * Host layer to identify this root hub. It is returned by the root hub driver
 * in the USB_HOST_DeviceEnumerate() function as the parent identifier when the
 * device is connected to the root hub. */

/* The hcdHandle is the driver handle. The hcdInterface pointer is of the type
 * DRV_USB_HOST_INTERFACE and points to the HCD interface. */

busObj->hcdInterface->rootHubInterface.rootHubInitialize( busObj->hcdHandle ,
                                                          rootHubDevice->deviceIdentifier );
busObj->hcdInterface->rootHubInterface.rootHubOperationEnable( busObj->hcdHandle , true );
```

When a device is attached, the Root Hub Driver will implement the required settling attach settling delay and will then call the USB Host Layer's USB_HOST_DeviceEnumerate function to enumerate the device. While calling this function, the root hub driver will provide the identifier that was provided to it in its initialize function. The USB_HOST_DeviceEnumerate function will return an identifier which uniquely identifies the attached device. The root hub driver uses this value to identify the device to the Host when the USB_HOST_DeviceDenumerate function is called on device detach. The following code example shows how the Root Hub driver calls the USB_HOST_DeviceEnumerate and the USB_HOST_DeviceDenumerate functions.

```
/* The following code shows how the root hub driver calls the
 * USB_HOST_DeviceEnumerate() function in the device attach interrupt. As seen
 * here, the root hub returns the identifier that the USB Host Layer assigned to
 * it the rootHubInitialize function call. The pUSBDrvObj->usbHostDeviceInfo
 * variable contains this identifier. */

if(PLIB_USB_InterruptFlagGet(usbID, USB_INT_ATTACH))
{
    /* We can treat this as a valid attach. We then clear the
     * detach flag and enable the detach interrupt. We enable
     * the Transaction interrupt */

    PLIB_USB_InterruptFlagClear(usbID, USB_INT_HOST_DETACH);
    PLIB_USB_InterruptEnable(usbID, USB_INT_HOST_DETACH);
    PLIB_USB_InterruptEnable(usbID, USB_INT_TOKEN_DONE);

    /* Ask the Host layer to enumerate this device. While calling
     * this function, the UHD of the parent device which is the
     * root hub in this case.
     * */
    pUSBDrvObj->attachedDeviceObjHandle = USB_HOST_DeviceEnumerate
                                          (pUSBDrvObj->usbHostDeviceInfo, 0);
}

/* The following code example shows how the root hub driver calls the
 * USB_HOST_DeviceDenumerate() function in the device detach interrupt. Note how
 * the attachedDeviceObjHandle that was assigned at the time of device
 * enumeration is returned to the Host Layer to let the Host know which device
 * is being detached. */

if((usbInterrupts & USB_INT_HOST_DETACH) && (enabledUSBInterrupts & USB_INT_HOST_DETACH))
{
    /* Perform other detach related handling */

    /* Ask the Host Layer to de-enumerate this device. */
    USB_HOST_DeviceDenumerate (pUSBDrvObj->attachedDeviceObjHandle);

    /* Disable the LS Direct Connect. It may have been enabled if the last
     attach was for a Low-Speed device. */
    PLIB_USB_EP0LSDirectConnectDisable(pUSBDrvObj->usbID);

    /* Continue to perform detach handling */
}
```

Root Hub Port Operation

The HCD Root Hub Driver exposes a set of port related functions that allow the USB Host Layer to control the port. The most commonly used functions are the function to reset the port and get the port speed. In this case, this is the speed of the attached device. The following code example shows how the USB Host Layer calls the hubPortReset, hubPortResetIsComplete and hubPortSpeedGet port functions.

```
/* The following code shows an example of how the Host Layer called the
 * hubPortReset function to reset the port to which the device is connected.
 * The code proceeds with the port reset if no device on the bus is in an
 * enumeration state. It will then call the hubPortReset function of the parent
 * hub of the device. The parent hub, hubInterface member of deviceObj points to
 * this driver, can be the root hub or an external hub */

if(!busObj->deviceIsEnumerating)
{
    /* Remember which device is enumerating */
    busObj->enumeratingDeviceIdentifier = deviceObj->deviceIdentifier;

    /* Grab the flag */
    busObj->deviceIsEnumerating = true;

    /* Reset the device */
    deviceObj->hubInterface->hubPortReset( deviceObj->hubHandle, deviceObj->devicePort );
}

/* The following code example shows how the Host checks if the port reset
 * operation has completed. If the reset operation has completed, the speed of
 * the attached device can be obtained. The reset settling delay can then be
 * started. */

 case USB_HOST_DEVICE_STATE_WAITING_FOR_RESET_COMPLETE:

    /* Check if the reset has completed */
    if(deviceObj->hubInterface->hubPortResetIsComplete
                ( deviceObj->hubHandle ,deviceObj->devicePort ))
    {
        /* The reset has completed. We can also obtain the speed of the
         * device. We give a reset recovery delay to the device */

        deviceObj->speed = deviceObj->hubInterface->hubPortSpeedGet
                           (deviceObj->hubHandle, deviceObj->devicePort);

        deviceObj->deviceState = USB_HOST_DEVICE_STATE_START_RESET_SETTLING_DELAY;
    }
```

Opening and Closing a Pipe

The HCD client can open a pipe to the device after resetting the device. The USB Host Layer calls the hostPipeSetup function in the DRV_USB_HOST_INTERFACE structure to open a pipe. The USB Host Layer must open a pipe to communicate to a specific endpoint on a target device. While opening the pipe, the USB Host Layer must specify parameters which specify the address of the target device, the type of the transfer that the pipe must support and the speed of the pipe. If the device is connected to a hub, the address of the hub must be specified. The HCD Pipe Setup function *is not* interrupt-safe. It should not be called in any event handler that executes in an interrupt context.

The Pipe Setup function returns a valid pipe handle if the pipe was opened successfully. Pipe creation may fail if the target device was disconnected or if there are insufficient resources to open the pipe. The pipe handle is then used along with the hostIRPSubmit function to transfer data between the Host and the device. The following code shows example usage of a Pipe Open function.

```
/* The following code example shows how the Host Layer uses the hostPipeSetup
 * function to open a control pipe to the attached device. Most of the
 * parameters that are passed to this function become known when the device is
 * attached. The pipe handle is checked for validity after the hostPipeSetup
 * function call. */

if(busObj->timerExpired)
{
    busObj->busOperationsTimerHandle = SYS_TMR_HANDLE_INVALID;
    /* Settling delay has completed. Now we can open default address
     * pipe and and get the configuration descriptor */

    SYS_DEBUG_PRINT(SYS_ERROR_INFO,
                    "\r\nUSB Host Layer: Bus %d Device Reset Complete.", busIndex);
```

```
    deviceObj->controlPipeHandle =
              deviceObj->hcdInterface->hostPipeSetup( deviceObj->hcdHandle,
              USB_HOST_DEFAULT_ADDRESS , 0 /* Endpoint */,
              deviceObj->hubAddress /* Address of the hub */,
              deviceObj->devicePort /* Address of the port */,
              USB_TRANSFER_TYPE_CONTROL, /* Type of pipe to open */
              0 /* bInterval */, 8 /* Endpoint Size */, deviceObj->speed );

    if(DRV_USB_HOST_PIPE_HANDLE_INVALID == deviceObj->controlPipeHandle)
    {
        /* We need a pipe else we cannot proceed */
        SYS_DEBUG_PRINT(SYS_ERROR_DEBUG,
        "\r\nUSB Host Layer: Bus %d Could not open control pipe. Device not supported.", busIndex);
    }
}
```

An open pipe consumes computational and memory resources and must therefore must be closed if it will not be used. This is especially true of pipes to a device that is detached. The Host Layer calls the hostPipeClose function in the DRV_USB_HOST_INTERFACE structure to close the pipe. The pipe to be closed is specified by the pipe handle. The Pipe Close function can be called from an event handler. It is interrupt safe. Closing a pipe will cancel all pending transfers on that pipe. The IRP callback for such canceled transfers will be called with the status USB_HOST_IRP_STATUS_ABORTED. The following code example shows an example of closing the pipe.

```
/* The following code example shows an example of how the Host Layer calls the
 * hostPipeClose function to close an open pipe. Pipe should be closed if it
 * will not used. An open pipe consumes memory resources. In this example, the
 * Host Layer closes the pipe if it was not able successfully submit an IRP to
 * this pipe. */

/* Submit the IRP */
if(USB_ERROR_NONE != deviceObj->hcdInterface->hostIRPSubmit
                  ( deviceObj->controlPipeHandle, & (deviceObj->controlTransferObj.controlIRP)))
{
    /* We need to be able to send the IRP. We move the device to
     * an error state. Close the pipe and send an event to the
     * application. The assigned address will be released when
     * the device in unplugged. */

    SYS_DEBUG_PRINT(SYS_ERROR_DEBUG,
        "\r\nUSB Host Layer: Bus %d Set Address IRP failed. Device not supported.", busIndex);

    /* Move the device to error state */
    deviceObj->deviceState = USB_HOST_DEVICE_STATE_ERROR;

    /* Close the pipe as we are about mark this device as unsupported. */
    deviceObj->hcdInterface->hostPipeClose(deviceObj->controlPipeHandle);
}
```

Transferring Data to an Attached Device

The USB Host Layer, the HCD client, needs to transfer data to the attached device to understand the device capabilities and to operate the device. The HCD uses a concept of Input Output Request Packet (IRP) to transfer data to and from the attached device. IRPs are transported over pipes which are setup by calling the USB Driver Pipe Setup function.

A Host IRP is a USB_HOST_IRP type data structure. The IRP is created by the Host layer and submitted to the HCD for processing through the hostIRPSubmit function. At the time of submitting the IRP, the pipe over which the IRP must be transported is specified. The data request in the IRP is transported using the attributes of pipe. When an IRP is submitted to the HCD, it is owned by the HCD and cannot be modified by the Host Layer until the HCD issues an IRP callback. The HCD will issue the IRP callback when it has completed or terminated processing of the IRP.

An IRP does not have its own transfer type. It inherits the properties of the pipe to which it is submitted. Hence an IRP becomes a control transfer IRP it was submitted to a control transfer pipe. A pipe allows multiple IRPs to be queued. This allows the Host Layer to submit IRPs to a pipe even while an IRP is being processed on the pipe. The HCD will process an IRP in the order that it was received. The following code example shows the USB_HOST_IRP data structure.

```
/* The following code example shows the USB_HOST_IRP structure. The Host Layer
 * uses this structure to place data transfer requests on a pipe. */

typedef struct _USB_HOST_IRP
{
    /* Points to the 8 byte setup command packet in case this is a IRP is
     * scheduled on a CONTROL pipe. Should be NULL otherwise */
    void * setup;

    /* Pointer to data buffer */
    void * data;
```

```
    /* Size of the data buffer */
    unsigned int size;

    /* Status of the IRP */
    USB_HOST_IRP_STATUS status;

    /* Request specific flags */
    USB_HOST_IRP_FLAG flags;

    /* User data */
    uintptr_t userData;

    /* Pointer to function to be called when IRP is terminated. Can be NULL, in
     * which case the function will not be called. */
    void (*callback)(struct _USB_HOST_IRP * irp);

    /*************************************
     * These members of the IRP should not be
     * modified by client
     *************************************/
   uintptr_t privateData[7];

} USB_HOST_IRP;
```

The `setup` member of the USB_HOST_IRP structure must point to the 8 byte setup packet for control transfers. The driver will send this 8 byte data in the Setup phase of the control transfer. It can be NULL for non-control transfers. This member is only considered if the IRP is submitted to a control transfer pipe. It is ignored for non-control transfer pipes. The structure of the setup command should match that specified in the USB 2.0 specification.

The `data` member of the USB_HOST_IRP structure points to a data buffer. This data buffer will contain the data that needs to be sent to the device for data stage of a OUT transfer, or it will contain the data that was received from the device during an IN transfer. Any hardware specific cache coherency and address alignment requirements must be considered while allocating this data buffer. The Driver Client should not modify or examine the contents of the IRP after the IRP has been submitted and is being processed. It can be examined after the driver has released the IRP.

The `size` member of the USB_HOST_IRP structure contains the size of the transfer. for Bulk transfers, the size of the transfer can exceed the size of the transaction (which is equal to size of the endpoint reported by the device). The HCD in such a case will split up the transfer into transactions. This process does not require external intervention. For control transfers, the size of the transfer is specified in the setup packet (pointed to by the `setup` member of the USB_HOST_IRP structure). The driver will itself process the Setup, Data (if required) and Handshake stages of control transfer. This process again does not require external intervention. For interrupt and isochronous transfers, the size of transfer specified in the IRP cannot exceed the size of the transaction. If size is specified as 0, then the driver will send a zero length packet. The `size` parameter of the IRP is updated by the driver when IRP processing is completed. This will contain the size of the completed transfer.

The `status` member of the IRP provides the completion status of the IRP and should be checked only when the IRP processing has completed. This is indicated by the driver calling the IRP callback function. The IRP status is a USB_HOST_IRP_STATUS type. The following code example shows the different possible values of the `status` member and an example of submit a control transfer IRP.

```
/* The following code shows an example of how the Host Layer populates
 * the IRP object and then submits it. IRP_Callback function is called when an
 * IRP has completed processing. The status of the IRP at completion can be
 * checked in the status flag. The size field of the irp will contain the amount
 * of data transferred. */

void IRP_Callback(USB_HOST_IRP * irp)
{
    /* irp is pointing to the IRP for which the callback has occurred. In most
     * cases this function will execute in an interrupt context. The application
     * should not perform any hardware access or interrupt unsafe operations in
     * this function. */

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            /* IRP was terminated due to an unknown error */
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            /* IRP was terminated by the application */
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            /* IRP was terminated due to a bus error */
```

```
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            /* IRP was terminated due to data error */
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            /* IRP was terminated because of a NAK timeout */
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            /* IRP was terminated because of a device sent a STALL */
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            /* IRP has been completed */
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            /* IRP has been completed but the amount of data processed was less
             * than requested. */
            break;

        default:
            break;
    }
}

/* In the following code example the a control transfer IRP is submitted to a
 * control pipe. The setup parameter of the IRP points to the Setup command of
 * the control transfer. The direction of the data stage is specified by the
 * Setup packet. */

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);

switch(result)
{
    case USB_ERROR_NONE:
        /* The IRP was submitted successfully */
        break;

    case USB_ERROR_HOST_PIPE_INVALID:
        /* The specified pipe handle is not valid */
        break;

    case USB_ERROR_OSAL_FUNCTION:
        /* An error occurred while trying to grab mutex */
        break;

    default:
        break;
}
```

The `flags` member of the USB_HOST_IRP structure specifies flags which affect the behavior of the IRP. The USB_HOST_IRP_FLAG enumeration specifies the available option. The USB_HOST_IRP_FLAG_SEND_ZLP causes the driver to add a Zero Length Packet (ZLP) to the data stage of the transfer when the transfer size is an exact multiple of the endpoint size. The USB_HOST_IRP_WAIT_FOR_ZLP flag will cause the driver to wait for a ZLP from the device in a case where the size of data received thus far in the transfer is an exact multiple of the endpoint

size.

The `callback` member of the USB_HOST_IRP structure points to a function which the driver calls when the IRP processing is completed. The Driver Client must implement this function and assign the pointer to this function to the `callback` member of the IRP. Every IRP can have its own callback function or one common callback function could be used. The callback function will execute in an interrupt context. The Driver Client should not execute interrupt unsafe, blocking, or computationally intensive operations in the callback function. The client can call hostIRPSubmit function in the IRP callback function to submit another IRP or resubmit the same IRP. The client can check the status and size of the IRP in the callback function.

The `userData` member of the USB_HOST_IRP structure can be used by the client to associate a client specific context with the Host. This context can then be used by the client, in the IRP callback function to identify the context in which the IRP was submitted. This member is particularly useful if the client wants to implement one callback function for all IRPs.

The `privateData` member of the IRP is used by the driver and should not be accessed or manipulated by the Driver Client. The following code examples show usage of IRPs to transfer data between the Host and the attached device and along with the different flags.

```
/* The following code shows an example of submitting an IRP to send data
 * to a device. In this example we will request the driver to send a ZLP after
 * sending the last transaction. The driver will send the ZLP only if the size
 * of the transfer is a multiple of the endpoint size. This is not a control
 * transfer IRP. So the setup field of the IRP will be ignored.  */

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE bulkOUTPipeHandle;
uint8_t data[128];

irp.data = data;
irp.size = 128;
irp.flags = USB_HOST_IRP_FLAG_SEND_ZLP ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IPRSubmit( bulkOUTPipeHandle, &irp );


/* The following code shows an example of submitting an IRP to receive
 * data to a device. In this example we will request the driver to wait for a
 * ZLP after receiving the last transaction. The driver will wait for the ZLP
 * only if the size of the transfer is a multiple of the endpoint size. This is
 * not a control transfer IRP. So the setup field of the IRP will be ignored.
 * */

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE bulkINPipeHandle;
uint8_t data[128];

irp.data = data;
irp.size = 128;
irp.flags = USB_HOST_IRP_FLAG_WAIT_FOR_ZLP ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IPRSubmit( bulkINPipeHandle, &irp );
```

USB Driver Device Mode Operation

Provides information on Device mode operation.

Description

The USB Driver operates can operate in the Device mode when it is initialized for Device mode or Dual Role operation. When operating in Device mode, the USB Driver is also referred to as the USB Controller Driver (USBCD). In Dual-Role mode, the USB Driver will switch to USBCD mode when the USB Driver Device Attach function is called.

The USB Driver Client must perform these steps to operate the USB Driver in Device mode.

1. Open the USB Driver to obtain the driver handle.
2. Set the event handler.
3. Wait for the application to attach the device to the bus.
4. Enable Endpoint 0 and respond to USB Host Enumeration requests.
5. Allow the application and function drivers to enable other endpoints and communicate with the Host.

The following sections discuss these operations in more detail.

General Device Mode Operations

Provides information on general Device mode operations.

Description

This section describes the USBCD operations such as setting event handlers and attaching and detaching the device.

Handling Device Mode Driver Events

The Device Layer will call the USBCD eventHandlerSet function to register the Device mode event handling function. The USBCD generates various events that indicate different states of the USB. These events are defined by the DRV_USB_EVENT enumeration. The following code example shows how the Device Layer registers the driver event handling function.

```
/* This code example shows the implementation of the USB_DEVICE_Attach and the
 * USB_DEVICE_Detach function. These functions are actually macro that map
 * directly deviceAttach and the deviceDetach function of the driverInterface
 * member of the deviceClient Object (which is the macro parameter) */

#define USB_DEVICE_Attach( x )  ((USB_DEVICE_OBJ *)x)->driverInterface->deviceAttach
                                ( ((USB_DEVICE_OBJ *)(x))->usbCDHandle)
#define USB_DEVICE_Detach( x )  ((USB_DEVICE_OBJ *)x)->driverInterface->deviceDetach
                                ( ((USB_DEVICE_OBJ *)x)->usbCDHandle)
```

If the driver is operating in interrupt mode, the client event handling function will execute in an interrupt context. The client should not call interrupt unsafe, computationally intensive or blocking functions in the event handler. The following code shows a small example of the Device Layer USBCD Event Handler:

```
/* This code example shows a partial implementation of the USB Device Layer
 * event handler. Note how the code type casts the referenceHandle parameter to
 * a USB_DEVICE_OBJ type. This referenceHandle is the same value that the Device
 * Layer passed when the event handler was set. This now easily allows one
 * implementation of the event handling code to be used by multiple Device
 * Layer instances. */

    void _USB_DEVICE_EventHandler
(
    uintptr_t referenceHandle,
    DRV_USB_EVENT eventType,
    void * eventData
)
{
    USB_DEVICE_OBJ* usbDeviceThisInstance;
    USB_DEVICE_MASTER_DESCRIPTOR * ptrMasterDescTable;
    USB_DEVICE_EVENT_DATA_SOF SOFFrameNumber;

    usbDeviceThisInstance = (USB_DEVICE_OBJ *)referenceHandle;

    /* Handle events, only if this instance is in initialized state */
    if( usbDeviceThisInstance->usbDeviceInstanceState <= SYS_STATUS_UNINITIALIZED )
    {
        /* The device should anyway not be attached when the Device Layer is
         * not initialized. If we receive driver event when the Device Layer is
         * not initialized, there is nothing we can do but ignore them. */
        return;
    }

    switch(eventType)
    {
        case DRV_USB_EVENT_RESET_DETECT:

            /* Clear the suspended state */
            usbDeviceThisInstance->usbDeviceStatusStruct.isSuspended = false;

            /* Cancel any IRP already submitted in the RX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointRx );

            /* Code not shown for the sake of brevity. */
```

```
    }
}
```

In the previous code example, the Device Layer (the Driver Client) sets the `hReferenceData` parameter, of the Event Handler Set function, to point to a local object. This pointer is returned to the Device Layer, in the event handler when an event occurs. For multiple instances of USB drivers in one application, this allows the Device Layer to easily associate a Device Layer specific context to the driver instance, thus simplifying implementation of the event handler.

Attaching and Detaching the Device

The USB Device Layer calls the USBCD deviceAttach and deviceDetach functions to attach and detach the device on the USB. The USB Device Layer should be ready to handle events which would occur when the device is attached on the bus. Hence the USB Device Layer should register the USBCD event handler before the attach function is called. The deviceAttach and deviceDetach functions can be called in an interrupt context. These functions are respectively called when the USB Device application detects a valid VBUS voltage and when the VBUS voltage is not valid.

Setting the Device Address

The USB Device Layer will call the USBCD deviceAddressSet function to set the USB address of the device. The Device Layer will do this when it receives the Set Address control request from the Host. The USBCD will reset the device address to '0' when it has received reset signaling from the root hub. The following code example shows how the USB Device Layer calls this function.

```
/* The following code example shows how the USB Device Layer calls the
 * DRV_USB_DEVICE_AddressSet function to set the address. The
 * DRV_USB_DEVICE_AddressSet function is actually a macro that calls the
 * deviceAddressSet function of the driverInterface of usbDeviceThisInstance
 * object. The usbDeviceThisInstance is Device Layer object.
 *
 * As seen in this code, the Device Layer calls the address set function when
 * the it a pending set address control request from the Host has completed. */


void _USB_DEVICE_Ep0TransmitCompleteCallback(USB_DEVICE_IRP * handle)
{
    USB_DEVICE_IRP * irpHandle = (USB_DEVICE_IRP *)handle;
    USB_DEVICE_OBJ * usbDeviceThisInstance;
    USB_DEVICE_CONTROL_TRANSFER_STRUCT * controlTransfer;

    usbDeviceThisInstance = (USB_DEVICE_OBJ *)irpHandle->userData;
    controlTransfer = &(usbDeviceThisInstance->controlTransfer);

    if(irpHandle->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        return;
    }

    if(usbDeviceThisInstance->usbDeviceStatusStruct.setAddressPending)
    {
        DRV_USB_DEVICE_AddressSet(usbDeviceThisInstance->usbCDHandle,
                                  usbDeviceThisInstance->deviceAddress);
        usbDeviceThisInstance->usbDeviceStatusStruct.setAddressPending = false;
    }

    /* Code not shown for the sake of brevity */

}
```

Device Current Speed and SOF Number

The USB Device Layer will call the USBCD deviceCurrentSpeedGet function to know the speed at which the device is attached to the USB. This allows the Device Layer to select the correct endpoint settings at the time of processing the Set Configuration request issued by the Host. The USB Device Layer will call the deviceSOFNumberGet function to return the SOF number at the time of the SOF event.

Device Remote Wake-up

The USB Device Layer will call the USBCD deviceRemoteWakeupStop and deviceRemoteWakeupStart functions to stop and start remote signaling. The Device layer application will call the USB Device Layer Stop and Start Remote Wakeup Signaling functions to remotely let the root hub know that the device is ready to be woken up. The timing of the remote signaling is controlled by the Device Layer. The client should call the remote wakeup function only when the device is suspended by the Host.

Device Endpoint Operations

Provides information on Device Endpoint operations.

Description

The UBSCD Endpoint functions allow the Driver Client to enable, disable, stall and clear the stall condition on an endpoint. The client submits requests to transmit and receive data from the USB Host on an endpoint.

Endpoint Enable and Disable functions

The USBCD client must enable an endpoint it must use the endpoint for communicating with the USB Host. The client will call the USBCD deviceEndpointEnable function to enable the endpoint. While calling this function, the client must specify the endpoint address, the transfer type to be processed on this endpoint and the maximum size of a transaction on this endpoint. This function is thread-safe when called in an RTOS application. The USBCD allows an endpoint to be accessed by one thread only. The USB Device Layer and the device function drivers will enable the endpoint when the Host sets the device configuration. The USBCD deviceEndpointIsEnabled function is available to check if an endpoint is enabled. The following code example shows how the USB Device Layer enables the device endpoint.

```
/* The following code example shows the USB Device Layer enables Endpoint 0 to
 * prepare for the enumeration process after it has received reset signaling
 * from the Host. The Device Layer calls the deviceEndpointEnable function to
 * to enable the endpoint. The driverInterface member of the
 * usbDeviceThisInstance structure points to the USB Device Mode Driver Common
 * Interface. */

void _USB_DEVICE_EventHandler
(
    uintptr_t referenceHandle,
    DRV_USB_EVENT eventType,
    void * eventData
)
{
    /* Code not shown due to space constraints */

    switch(eventType)
    {
        case DRV_USB_EVENT_RESET_DETECT:

            /* Clear the suspended state */
            usbDeviceThisInstance->usbDeviceStatusStruct.isSuspended = false;

            /* Cancel any IRP already submitted in the RX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointRx );

            /* Cancel any IRP already submitted in the TX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointTx );

            /* Deinitialize all function drivers.*/
            _USB_DEVICE_DeInitializeAllFunctionDrivers ( usbDeviceThisInstance );

            /* Disable all endpoints except for EP0.*/
            DRV_USB_DEVICE_EndpointDisableAll(usbDeviceThisInstance->usbCDHandle);

            /* Enable EP0 endpoint in RX direction */
            (void)usbDeviceThisInstance->driverInterface->deviceEndpointEnable
                  (usbDeviceThisInstance->usbCDHandle,
                    controlEndpointTx, USB_TRANSFER_TYPE_CONTROL, USB_DEVICE_EP0_BUFFER_SIZE);

            /* Code not shown due to space constraints */

            break;
    }
}
```

The USB Device Layer and the Function drivers will disable an endpoint when the Host sets a zero-device configuration or when the Host resets the device. The USBCD deviceEndpointDisable function disables an endpoint. When an endpoint is disabled, it does not accept requests for Host communication. Disabling an endpoint does not cancel any communication requests that that have been submitted on the endpoint. These requests must be canceled explicitly.

Device Endpoint Stall and Stall Clear

The USBCD client can call the deviceEndpointStall and deviceEndpointStallClear functions to stall and cleat the stall on an endpoint respectively. The USB Device Layer and function driver may stall endpoint to indicate error or to indicate a protocol state. The endpoint stall condition may be

cleared in response to a USB Host Clear Feature request. Stalling or clearing the stall on an endpoint will cause all communication requests on the endpoint to be canceled. The function calls are thread safe and interrupt safe. The deviceEndpointIsStalled function is also available to check if an endpoint is in a stalled state. The following code example shows how the USB Device Layer calls these functions to stall and clear the stall on an endpoint.

```
/* The following code example shows how the USB Device Layer calls the driver
 * endpoint stall function (deviceEndpointStall) to stall an endpoint when the a
 * Host send a Set Feature request with feature selector set to endpoint halt.
 * The endpoint to be halted is identified in the setup packet and is identified
 * in this code example as usbEndpoint. Also shown is how the stall clear
 * (deviceEndpointStallClear) and stall status check (deviceEndpointIsStalled)
 * functions are called. */

/* The driverInterface member of the usbDeviceThisInstance structure is a
 * pointer to the USB Driver Common Interface. */

void _USB_DEVICE_ProcessStandardEndpointRequest
(
    USB_DEVICE_OBJ * usbDeviceThisInstance,
    uint8_t interfaceNumber,
    USB_SETUP_PACKET * setupPkt
)
{
    USB_ENDPOINT usbEndpoint;
    usbEndpoint = setupPkt->bEPID;

    if( setupPkt->bRequest == USB_REQUEST_GET_STATUS )
    {
        usbDeviceThisInstance->getStatusResponse.status = 0x00;
        usbDeviceThisInstance->getStatusResponse.endPointHalt
            =  usbDeviceThisInstance->driverInterface->deviceEndpointIsStalled
               (usbDeviceThisInstance->usbCDHandle, usbEndpoint );

        USB_DEVICE_ControlSend( (USB_DEVICE_HANDLE)usbDeviceThisInstance,
                (uint8_t *)&usbDeviceThisInstance->getStatusResponse, 2 );
    }
    else if( setupPkt->bRequest == USB_REQUEST_CLEAR_FEATURE )
    {
        if( setupPkt->wValue == USB_FEATURE_SELECTOR_ENDPOINT_HALT )
        {
            usbDeviceThisInstance->driverInterface->deviceEndpointStallClear
            (usbDeviceThisInstance->usbCDHandle, usbEndpoint );
            USB_DEVICE_ControlStatus((USB_DEVICE_HANDLE)usbDeviceThisInstance,
                                     USB_DEVICE_CONTROL_STATUS_OK );
        }
    }
    else if (setupPkt->bRequest == USB_REQUEST_SET_FEATURE )
    {
        if( setupPkt->wValue == USB_FEATURE_SELECTOR_ENDPOINT_HALT )
        {
            usbEndpoint = setupPkt->bEPID;
            usbDeviceThisInstance->driverInterface->deviceEndpointStall
                                 (usbDeviceThisInstance->usbCDHandle, usbEndpoint );
            USB_DEVICE_ControlStatus((USB_DEVICE_HANDLE)usbDeviceThisInstance,
                                     USB_DEVICE_CONTROL_STATUS_OK );
        }
    }

    /* Additional code is not shown due to space constraints */
}
```

Transferring Data to the Host

Provides information on transferring data to the Host.

Description

The USB Device Layer, the USBCD client, needs to transfer data to the Host in response to enumeration requests for general operation on the device. The USB uses a concept of Input Output Request Packet (IRP) to transfer data to and from the Host. IRPs are transported over endpoints which are enabled by calling the USBCD Endpoint Enable function.

A Device IRP is a USB_DEVICE_IRP type data structure. The IRP is created by the Device Layer and submitted to the USBCD for processing through the deviceIRPSubmit function. At the time of submitting the IRP, the endpoint over which the IRP must be transported is specified. The data request in the IRP is transported using the attributes of the endpoint. When an IRP is submitted to the USBCD, it is owned by the USBCD and cannot be modified by the Device Layer until the USBCD issues an IRP callback. The USBCD will issue the IRP callback when it has completed or terminated processing of the IRP.

An IRP does not have its own transfer type. It inherits the properties of the endpoint to which it is submitted. Hence an IRP becomes a control transfer IRP it was submitted to a control endpoint. An endpoint allows multiple IRPs to be queued. This allows the Device Layer to submit IRPs to an endpoint even while an IRP is being processed on the endpoint. The USBCD will process an IRP in the order that it was received. The following code example shows the USB_DEVICE_IRP data structure:

```
/* This code example shows the USB_DEVICE_IPR structure. The Device Layer
 * uses such a structure to transfer data through the driver. A structure of
 * this type is allocated by the Device Layer and the other function drivers and
 * passed to the deviceIRPSubmit function. */

typedef struct _USB_DEVICE_IRP
{
    /* Pointer to the data buffer */
    void * data;

    /* Size of the data buffer */
    unsigned int size;

    /* Status of the IRP */
    USB_DEVICE_IRP_STATUS status;

    /* IRP Callback. If this is NULL, then there is no callback generated */
    void (*callback)(struct _USB_DEVICE_IRP * irp);

    /* Request specific flags */
    USB_DEVICE_IRP_FLAG flags;

    /* User data */
    uintptr_t userData;

    /**********************************
     * The following members should not
     * be modified by the client
     **********************************/
    uint32_t privateData[3];

} USB_DEVICE_IRP;
```

The `data` member of the USB_DEVICE_IRP structure points to a data buffer. This data buffer will contain the data that needs to be sent to the Host for the data stage of an IN transfer. For an OUT transfer, it will contain the data that was received from the Host. Any hardware specific cache coherency and address alignment requirements must be considered while allocating this data buffer. The Driver Client should not modify or examine the contents of the IRP after the IRP has been submitted and is being processed. It can be examined after the driver has released the IRP.

The `size` member of the USB_DEVICE_IRP structure specifies the size of the data buffer. The transfer will end when the device has sent or received size number of bytes. While sending data to the Host, the IRP size can exceed the size of the transaction (which is equal to the size of the endpoint). The USBCD in such a case will split up the transfer into transactions. This process does not require external intervention. The driver uses receive and transmit IRPs to process control transfers. When the driver receives a Setup packet, the IRP completion status would be USB_DEVICE_IRP_STATUS. The Driver Client should then use additional receive and transmit IRPs to complete the control transfer.

For interrupt and isochronous transfers, the size of transfer specified in the IRP cannot exceed the size of the transaction. If size is specified as 0, then the driver will send or expect a zero length packet. The `size` parameter of the IRP is updated by the driver when IRP processing is completed. This will contain the size of the completed transfer.

The `status` member of the IRP provides the completion status of the IRP and should be checked only when the IRP processing has completed. This is indicated by the driver calling the IRP callback function. The IRP status is a USB_DEVICE_IRP_STATUS type. The following code example shows the different possible values of the `status` member and example usage of IRPs to transfer data between the device and the Host.

```
/* The followoing code shows example usage of the device IRP. The submit status
 * of the IRP is available when IRP submit function returns. The completion
 * status of the IRP is available when the IRP has terminated and the IRP
 * callback function is invoked. The IRP callback
 * function shown in this example shows the possible complete status of the IRP.
 * The end application may or may not handle all the cases. Multiple IRPs can be
 * queued on an endpoint. */

void IRP_Callback(USB_DEVICE_IRP * irp)
{
```

```
    /* irp is pointing to the IRP for which the callback has occurred. In most
     * cases this function will execute in an interrupt context. The application
     * should not perform any hardware access or interrupt unsafe operations in
     * this function. */

    switch(irp->status)
    {
        case USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST:
            /* The IRP was aborted because the Host cleared the stall on the
             * endpoint */
            break;

        case USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT:
            /* IRP was aborted because the endpoint halted */
            break;


        case USB_DEVICE_IRP_STATUS_ABORTED:
            /* USB Device IRP was aborted by the function driver */
            break;

        case USB_DEVICE_IRP_STATUS_ERROR:
            /* An error occurred on the bus when the IRP was being processed */
            break;

        case USB_DEVICE_IRP_STATUS_COMPLETED:
            /* The IRP was completed */
            break;

        case USB_DEVICE_IRP_STATUS_COMPLETED_SHORT:
            /* The IRP was completed but the amount of data received was less
             * than the requested size */
            break;

        default:
            break;

    }
}

/* In the following example, the IRP is submitted to Endpoint 0x84. This is
 * interpreted as an IN direction endpoint (MSB of 0x84 is 1) and Endpoint 4.
 * The data contained in source will be sent to the USB Host. Assuming
 * the endpoint size is 64, the 130 bytes of data in this case will be sent to
 * the Host in three transaction of 64, 64 and 2 bytes. A transaction completes
 * when the Host polls (sends an IN token) the device.  The callback function
 * will then called indicating the completion status of the IRP. The application
 * should not modify the privateData field of the IRP. If the IRP was submitted
 * successfully, the buffer will be owned by the driver until the IRP callback
 * function has been called. Because the size of the transfer is not a multiple
 * of the endpoint size, the IRP flag must be set
 * USB_DEVICE_IRP_FLAG_DATA_COMPLETE. This directs the driver to not perform any
 * explicit signaling to the Host to indicate end of transfer. The last packet
 * in this case is a short packet and this signals the end of the transfer. */

USB_DEVICE_IRP irp;
USB_ERROR result;
uint8_t source[130];

irp.data = source;
irp.size = 130;
irp.called = IRP_Callback;
flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
userData = &someApplicationObject;

result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);

switch(result)
{
```

```
        case USB_ERROR_PARAMETER_INVALID:
            /* This can happen if the driverHandle is invalid */
            break;

        case USB_ERROR_DEVICE_IRP_IN_USE:
            /* This can happen if the IRP is being resubmitted while it is still in
             * process (it was submitted before but processing has not completed */
            break;

        case USB_ERROR_DEVICE_ENDPOINT_INVALID;
            /* The endpoint to which this IRP is being submitted is not provisioned
             * in the system. This is controller by DRV_USBFS_ENDPOINTS_NUMBER
             * configuration parameter. */
            break;

        case USB_ERROR_ENDPOINT_NOT_CONFIGURED:
            /* The endpoint to which this IRP is being submitted is not enabled. It
             * must be enabled by calling the DRV_USBFS_DEVICE_EndpointEnable()
             * function. */
            break;

        case USB_ERROR_PARAMETER_INVALID:
            /* The USB_DEVICE_IRP_FLAG_DATA_PENDING flag was specified but the
             * transfer size is not a multiple of the endpoint size. If the IRP was
             * submitted to a receive endpoint, this error can occur if the size is
             * not a multiple of the endpoint size. */
            break;

        case USB_ERROR_OSAL_FUNCTION:
            /* An error occurred while trying to grab a mutex. This is applicable
             * when the driver is running with a RTOS. */
            break;

        case USB_ERROR_NONE:
            /* The IRP was submitted successfully. */
            break;

        default:
            break;
    }

    /* The following code example shows how an IRP is submitted to an OUT endpoint.
     * In this case data will be pointing to a buffer where the received data will
     * be stored. Note that the size of the IRP should be a multiple of the endpoint
     * size. The flags parameter is ignored in the data receive case. The IRP
     * terminates when the specified size of bytes has been received (the Host sends
     * OUT packets) or when a short packet has been received. */

    USB_DEVICE_IRP irp;
    USB_ERROR result;
    uint8_t destination[128];

    irp.data = destination;
    irp.size = 128;
    irp.called = IRP_Callback;
    userData = &someApplicationObject;

    result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x04, &irp);
```

For IRPs submitted to an Interrupt or Isochronous endpoints, the driver will always send either the less than or equal to the maximum endpoint packet size worth of bytes in a transaction. The application could either submit an IRP per Interrupt/Isochronous polling interval or it could submit one IRP for multiple polling intervals.

The `flags` member of the USB_DEVICE_IRP structure specifies flags which affect the behavior of the IRP. The USB_DEVICE_IRP_FLAG enumeration specifies the available option. The USB_DEVICE_IRP_FLAG_DATA_COMPLETE causes the driver to add a Zero Length Packet (ZLP) to the data stage of the IN transfer when the transfer size is an exact multiple of the endpoint size. If the transfer size is not a multiple of the endpoint size, no ZLP will be sent. The USB_DEVICE_IRP_FLAG_PENDING flag will cause the driver to not send a ZLP in a case where the size of the IN transfer is an exact multiple of the endpoint size. The following code example demonstrates this.

```
/* In the following code example, the IRP is submitted to an IN endpoint whose size
 * is 64. The transfer size is 128, which is an exact multiple of the endpoint
```

```
 * size. The flag is set to USB_DEVICE_IRP_FLAG_DATA_COMPLETE. The driver
 * will send two transactions of 64 bytes each and will then automatically send a
 * Zero Length Packet (ZLP), thus completing the transfer. The IRP callback will
 * be invoked when the ZLP transaction has completed. */

USB_DEVICE_IRP irp;
USB_ERROR result;
uint8_t source[128];

irp.data = source;
irp.size = 128;
irp.called = IRP_Callback;
flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
userData = &someApplicationObject;

result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);

/* In the following code example, the IRP is submitted to an IN endpoint whose size
 * is 64. The transfer size is 128, which is an exact multiple of the endpoint
 * size. The flag is set to to USB_DEVICE_IRP_FLAG_DATA_PENDING. The driver will
 * send two transactions of 64 bytes each but will not send a ZLP. The USB Host
 * can then consider that there is more data pending in the transfer. The IRP
 * callback will be invoked when the two transactions have completed. */

USB_DEVICE_IRP irp;
USB_ERROR result;
uint8_t source[128];

irp.data = source;
irp.size = 128;
irp.called = IRP_Callback;
flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
userData = &someApplicationObject;

result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);
```

The `callback` member of the USB_DEVICE_IRP structure points to a function which the driver calls when the IRP processing is completed. The Driver Client must implement this function and assign the pointer to this function to the `callback` member of the IRP. Every IRP can have its own callback function or one common callback function could be used. The callback function will execute in an interrupt context. The Driver Client should not execute interrupt unsafe, blocking or computationally intensive operations in the callback function. The client can call deviceIRPSubmit function in the IRP callback function to submit another IRP or resubmit the same IRP. The client can check the status and size of the IRP in the callback function.

The `userData` member of the USB_DEVICE_IRP structure can be used by the client to associate a client specific context with the Host. This context can then be used by the client, in the IRP callback function to identify the context in which the IRP was submitted. This member is particularly useful if the client wants to implement one callback function for all IRPs.

The `privateData` member of the IRP is used by the driver and should not be accessed or manipulated by the Driver Client.

PIC32MX USB Driver

Provides information on the USB Driver specific to PIC32MX devices.

Description

The PIC32MX USB Driver in MPLAB Harmony provides API functions that allow the MPLAB Harmony USB Host and Device Stack to access the USB while operating on a PIC32MX microcontroller. The driver implements the USB Driver Common Interface required by the USB Host and Device Stack. It abstracts the USB module operational details from the Host and Device Stack and provides the stacks with a modular access mechanism to the USB. The PIC32MX USB Driver features the following:

- USB 2.0 Full Speed operation in Peripheral mode
- USB 2.0 Full Speed and Low Speed USB Peripheral Support in Host mode
- Designed for Dual Role Operation
- Capable of operating multiple USB modules
- Features non-blocking function and is interoperable with other MPLAB Harmony modules
- Features thread safe functions when operating within an RTOS
- Capable of operating in Polled and Interrupt modes
- Implements the USB Driver Common Interface required by the MPLAB Harmony USB Host and Device Stack
- Completely configurable through MPLAB Harmony Configurator (MHC) tool
- Implements feature separation (Host and Device mode functions are implemented across different files)

Note: This help section only discusses features that are unique to the PIC32MX USB Driver and are not a part of the USB Driver Common Interface. The driver functions that implement the USB Driver Common Interface are described in the Common Interface Help section.

While the PIC32MX USB module supports USB "On-The-Go" (OTG), this release of the PIC32MX Driver does not implement USB OTG protocol support.

This help section only provides relevant information about the operation of the USB. The reader is encouraged to refer to the USB 2.0 Specification available at www.usb.org for a detailed explanation of USB protocol.

Using the Library

This topic describes the basic architecture of the PIC32MX USB Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usbfs.h`

The interface to the PIC32MX USB Driver library is defined in the `drv_usbfs.h` header file.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

Library Overview

Provides an overview of the library.

Description

The PIC32MX USB Driver will typically be used by a USB Host and/or Device Stack. The USB Host and Device Stack operate as driver client applications. The driver is initialized as part of the MPLAB Harmony System Initialization. The driver initialization data structure specifies the operation mode (Host, Device, or Dual Role) of the driver. The driver features task routines to be called in the MPLAB Harmony application tasks function (SYS_Tasks function) and the USB Module Interrupt Service Routine (ISR).

The Host and the Device Stack can open the driver only when initialization has completed. It will continue to return an invalid driver handle while the initialization is in progress. Once opened, the Device Mode function can be called if the driver is operating in Device mode. The Host Mode function can be called if the driver is operating in Host mode. In Dual Role operation mode, the driver supports Host and Device operation in the same application. Even then, the driver will either operate as a USB Host or Device. OTG operation is not supported.

The PIC32MX USB Driver features RTOS thread-safe functions. This allows the driver client application to safely call driver functions across different RTOS threads. Not all of the driver functions are interrupt-safe.

In addition to the USB Driver, which implements the USB Driver Common Interface, the PIC32MX USB Driver implements functions which are required for its operation in the MPLAB Harmony framework. The following table lists the different categories of functions in the PIC32MX USB Driver.

Library Interface Section	Description
System Function	These functions are accessed by the MPLAB Harmony System module. They allow the driver to be initialized, deinitialized and maintained. These functions are implemented in the `drv_usbfs.c` source file.
Client Core Functions	These functions allow the USB Host and Device Stack to open, close and perform other general driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs.c` source file.
Device Mode Operation Functions	These functions allow the USB Device Stack to perform USB Device mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_device.c` source file
Host Mode Operation Functions	These functions allow the USB Host Stack to perform USB Host mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_host.c` source file.
Root Hub Functions	These functions allow the USB Host Stack to access the driver Root hub operation. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_host.c` source file.

Abstraction Model

Provides information on the abstraction model for the library.

Description

The PIC32MX USB Driver implements the abstraction model defined by the USB Driver Common interface. This interface abstracts USB module specific details and provides a module independent interface to the driver client applications.

While operating in Device mode, the driver expects the client application (the USB Device Stack) to enable endpoints and then submit I/O request

packet (IRP) requests to the enabled endpoints. Multiple IRPs can be queued on an endpoint. The driver calls the IRP callback function when the IRP is processed. The driver allows the client application to also attach and detach the device on the bus. It generates events which indicate USB states.

While operating in Host mode, the driver expects the client application (the USB Host Stack) to open pipes to endpoints on the connected device. The client application can then submit IRPs to the pipes. Multiple IRPs can be queued on a pipe. The driver calls the IRP callback function when the IRP is processed. The driver will call application defined functions to enumerate and denumerate a device. These functions are called when the driver detect device attach and detach respectively. The driver also exports root hub functions to the client application. This allows the client application to treat the driver as a single port hub

Please refer to the PIC32 USB Driver Common Interface help section for more details on the driver abstraction model.

How the Library Works

Provides information on how the library works.

Description

This section only explains aspects of driver operation which are unique to the PIC32MX USB Driver. Major driver operations are described in the PIC32 USB Driver Common Interface help section.

Driver Initialization

Note: While generating a MPLAB Harmony USB project with MHC, the initialization code for the driver is generated automatically based on selections made in the USB Host stack or Device Stack Configuration trees.

The PIC32MX USB Driver must be initialized so that a client application can open. The client application will not be able to open the driver if the initialization is in progress or has failed. The driver is initialized by calling the DRV_USBFS_Initialize function. This function is called from the SYS_Initialize function in the MPLAB Harmony application project and accepts two input parameters. The index parameter defines the instance of the USB Driver to be initialized. This becomes significant when the PIC32MX microcontroller has more than one USB module. The init parameter is a driver specific data structure of the type DRV_USBFS_INIT. This structure is shown in the following code example.

```
/* This code snippet show the PIC32MX USB Driver Initialization data structure.
 * A structure of this type must be provided to the DRV_USBFS_Initialize()
 * function. */

typedef struct
{
    /* System Module Initialization */
    SYS_MODULE_INIT moduleInit;

    /* Identifies the USB peripheral to be used. This should be the USB PLIB
       module instance identifier. */
    uint8_t usbID;

    /* This should be set to true if the USB module must stop operation in IDLE
       mode */
    bool stopInIdle;

    /* This should be set to true if the USB module must suspend when the CPU
       enters sleep mode. */
    bool suspendInSleep;

    /* Specify the interrupt source for the USB module. This should be Interrupt
       PLIB Interrupt source identifier for the USB module instance specified in
       usbID. */
    INT_SOURCE interruptSource;

    /* Specify the operational speed of the USB module. This should always be
       set to USB_SPEED_FULL. The use of this parameter is deprecated. */
    USB_SPEED operationSpeed;

    /* Specify the operation mode of the USB module. This defines if the USB
     * module will support Device, Host or Dual Role operation */
    DRV_USBFS_OPMODES operationMode;

    /* A pointer to the endpoint descriptor table. This should be aligned at 512
       byte address boundary. The size of the table is equal to the
       DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed
       in the application. */
    void * endpointTable;
```

```
    /* Root hub available current in mA. This specifies the amount of current
       that root hub can provide to the attached device. This should be
       specified in mA. This is required when the driver is required to operate
       in host mode. */
    uint32_t rootHubAvailableCurrent;

    /* When operating in Host mode, the application can specify a Root Hub port
       enable function. This parameter should point to Root Hub port enable
       function. If this parameter is NULL, it implies that the Port is always
       enabled. */
    DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;

    /* When operating in Host mode, the application can specify a Root Port
       Indication. This parameter should point to the Root Port Indication
       function. If this parameter is NULL, it implies that Root Port Indication
       is not supported. */
    DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;

    /* When operating is Host mode, the application can specify a Root Port
       Overcurrent detection. This parameter should point to the Root Port
       Indication function. If this parameter is NULL, it implies that
       Overcurrent detection is not supported. */
    DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;

} DRV_USBFS_INIT;
```

The operationMode parameter defines by the driver operation mode. parameter in the initialization data structure. This can be set DRV_USBFS_OPMODE_DEVICE, DRV_USBFS_OPMODE_HOST or DRV_USBFS_OPMODE_DUAL_ROLE for device, host and dual role operation respectively.

The `endpointTable` parameter must point to a byte array. The size of the array depends on the maximum number of device endpoints that application needs. A direction of an endpoint counts as one endpoint. Each endpoint requires 32 bytes. Therefore, if the USB Device requires 3 endpoints (Endpoint 0 + Endpoint 1 IN + Endpoint 2 OUT), the size of the array will 96 bytes (32 * 3). The byte array start address must be located on a 512 byte boundary. When operating in host mode, the driver will use only one endpoint and size of the endpoint table array should be set to 32.

The `rootHubAvailableCurrent` parameter should be set to the maximum current that VBUS power supply can provide on the bus. The driver does not use this information directly. It provides this data to the client application while operating in host mode.

The `portPowerEnable` parameter must point to a Port Power Enable function. The driver, while operating in host mode, will call this function to enable the VBUS switch. This function should activate the VBUS switch if the driver calls this function with the enable parameter set to true. It should deactivate the switch if the driver calls this function with the enable parameter set to false. This parameter should be set to NULL if such a switch (of the switch control) is not available in the application.

The `portIndication` parameter must point to a Port Indication function. The driver, while operating in host mode, will call this function to indicate the current state of the port. The driver will call this function with LED color status as defined in the Chapter 11 of the USB 2.0 Specification. This parameter should be set to NULL if such a LED indication is not available in the application.

The `portOverCurrentDetect` parameter must point to a Port Overcurrent Detect function. The driver, while operating in Host mode, will call this function periodically to check if the attached device is overdrawing current. If the function should return true if such a condition exists. This parameter should be set to NULL if such detection is not available in the application.

The following code example shows initialization of the driver for device mode operation.

```
/* This code shows an example of DRV_USBFS_INIT data structure for
 * device mode operation. Here the driver is initialized to work with USB1 USB
 * module. Note how the endPointTable is defined. It should be aligned on a 512
 * byte boundary. */

DRV_USBFS_INIT init;
SYS_MODULE_OBJ usbDriverObj;

uint8_t __attribute__((aligned(512))) endPointTable[DRV_USBFS_ENDPOINTS_NUMBER * 32];

const DRV_USBFS_INIT drvUSBInit =
{
    /* Assign the endpoint table */
    .endpointTable = endPointTable,

    /* Interrupt Source for USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* System module initialization. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},
```

```
    /* Configure the driver for device mode operation. */
    .operationMode = DRV_USBFS_OPMODE_DEVICE,

    /* Configure the driver to operate at full speed. */
    .operationSpeed = USB_SPEED_FULL,

    /* Stop in idle */
    .stopInIdle = false,

    /* Suspend in sleep */
    .suspendInSleep = false,

    /* Identifies peripheral (PLIB-level) ID */
    .usbID = USB_ID_1
};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecasted to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBFS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBFS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBFS_Tasks() takes the driver module object returned by the
     * DRV_USBFS_Initialize funciton as a parameter. */

    DRV_USBFS_Tasks(usbDriverObj);
}

void __ISR(_USB_1_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBFS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBFS_Tasks_ISR(sysObj.drvUSBObject);
}
```

The following code example shows initialization of the driver for host mode operation.

```
/* This code shows an example of the USBFS driver can be configured for
 * host mode operation. For host mode operation, only one endpoint is needed and
 * hence the size of the endpoint table is 32 bytes (for one endpoint). In this
 * example, the BSP_USBVBUSSwitchOverCurrentDetect function checks for over
 * current condition and the BSP_USBVBUSPowerEnable function enables the VBUS
 * power. The port indication function is not implemented and hence the
 * portIndication member of the initialization data structure is set to NULL. */

/* The implementation of the port over current detect, indication and the VBUS
 * power supply functions is discussed later in this help section. */

uint8_t __attribute__((aligned(512))) endpointTable[32];

DRV_USBFS_INIT drvUSBFSInit =
{
    /* Pointer to the endpoint table */
    .endpointTable = endpointTable,

    /* Interrupt Source for the USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* This should always be set to SYS_MODULE_POWER_RUN_FULL. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},
```

```
    /* Configure for host mode operation. */
    .operationMode = DRV_USBFS_OPMODE_HOST,

    /* The driver should run at full speed. */
    .operationSpeed = USB_SPEED_FULL,

    /* Port indication function is not implemented. */
    .portIndication = NULL,

    /* This is the over current detect function. */
    .portOverCurrentDetect = BSP_USBVBUSSwitchOverCurrentDetect,

    /* This is the VBUS Power enable function */
    .portPowerEnable = BSP_USBVBUSPowerEnable,

    /* Here we state that the VBUS power supply can provide at most 500 mA of
     * current */
    .rootHubAvailableCurrent = 500,

    /* Module will operate in IDLE. */
    .stopInIdle = false,

    /* Module will not suspend automatically in sleep */
    .suspendInSleep = false,

    /* USB Module ID is 1 */
    .usbID = USB_ID_1
};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecasted to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBFS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBFS_Initialize(DRV_USBFS_INDEX_0,
                                        (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBFS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBFS_Tasks() takes the driver module object returned by the
     * DRV_USBFS_Initialize funciton as a parameter. */

    DRV_USBFS_Tasks(usbDriverObj);
}

void __ISR(_USB_1_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBFS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBFS_Tasks_ISR(sysObj.drvUSBObject);
}
```

The PIC32MX USB Driver requires definition of configuration constants to be available in the `system_config.h` file of the MPLAB Harmony Application Project Configuration. Refer to the Configuring the Library section for details.

Multi-client Operation

The PIC32MX USB Driver supports multi-client operation. In that, it can be opened by two application clients. This is required where Dual Operation is desired. The following should be noted when using multi-client operation:

- The driver should be initialized for Dual Role Operation mode.
- The DRV_USBFS_Open function can be called at the most twice in the application. The driver supports a maximum of two clients.

- A client can access either the host or device functionality of the driver. It cannot do both.
- It is possible for the two clients to operate in two different threads while operating with an RTOS.

Note: The typical the application clients for PIC32MX USB Driver would be the MPLAB Harmony USB Host and Device Stack. The complexity of operating the driver in Dual Role mode is handled by the stack operation. The MHC will configure the driver for Dual Role operation when such operation is selected in USB Stack configuration tree.

USB Driver Common Interface

The PIC32MX USB Driver exports its implementation of the USB Driver Common Interface to the Host and Device Layer via the DRV_USBFS_HOST_INTERFACE and DRV_USBFS_DEVICE_INTERFACE structures. The DRV_USBFS_HOST_INTERFACE structure is defined in the `drv_usbfs_host.c` file. The following code example shows this structure.

```
/*********************************************************
 * This structure is a set of pointer to the USBFS driver
 * functions. It is provided to the host and device layer
 * as the interface to the driver.
 * *******************************************************/

DRV_USB_HOST_INTERFACE gDrvUSBFSHostInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBFS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBFS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBFS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBFS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBFS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBFS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBFS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet = DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBFS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBFS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBFS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBFS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

The DRV_USBFS_DEVICE_INTERFACE structure is defined in the `drv_usbfs_device.c` file. The following code example shows this structure. The MPLAB Harmony USB Host and Device stack perform driver independent access through the function pointers contained in these structures.

```
/*************************************************
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * device layer when the device layer must use the
 * PIC32MX USB Controller.
 ************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBFSDeviceInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBFS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBFS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBFS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBFS_DEVICE_Attach,
    .deviceDetach = DRV_USBFS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBFS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBFS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBFS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBFS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBFS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBFS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBFS_DEVICE_IRPSubmit,
```

```
    .deviceIRPCancel = DRV_USBFS_DEVICE_IRPCancel,
    .deviceIRPCancelAll = DRV_USBFS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBFS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBFS_DEVICE_RemoteWakeupStart,
    .deviceTestModeEnter = NULL


};
```

Operation with RTOS

The PIC32MX USB Driver is designed to operate with a RTOS. The driver implementation uses the MPLAB Harmony Operating System Abstraction Layer (OSAL). This allows the driver to function with entire range of RTOSes supported in MPLAB Harmony. The following points must be considered while using the driver with an RTOS.

- The driver can be opened from different threads
- In Device mode, an enabled endpoint should only be accessed from one thread. For example, if an application requires two endpoints, Endpoint 2 and Endpoint 3, the application could contain two threads, one accessing Endpoint 2 and another accessing Endpoint 3. The thread accessing Endpoint 2 cannot access Endpoint 3.
- While operating in Host mode, endpoint pipes can be opened from different threads. A pipe handle to an open pipe cannot be shared across threads.

Host Mode Attach Detach Operation

When the PIC32MX USB Driver operating in Host mode detects a device attach or detach, it implements debouncing before signaling attach detach signal to the USB Host Stack. When the device is attached, the driver waits for DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION milliseconds to allow for the mechanical chatter, which occurs when the device is inserted into the host receptacle, to settle. If the device is still attached after the DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION expires, the driver calls the USB_HOST_DeviceEnumerate function to let the host stack enumerate the device. It also starts checking for Device Detach.

When the device is detached, the driver waits for DRV_USBFS_POST_DETACH_DELAY milliseconds to allow for the detach operation to settle. If the device is indeed detached after the DRV_USBFS_POST_DETACH_DELAY delay expires, the driver calls USB_HOST_DeviceDenumerate function to let the USB Host stack denumerate the device. It then starts checking for device attach.

Root Hub Operation

The PIC32MX USB Driver implements a Root Hub Driver Interface. This allows the driver to emulate a hub. The USB Host Stack enumerates the Root Hub as a device. The Host Stack then does not differentiate between an external hub and the root hub. While emulating a hub, the PIC32MX USB Driver Root Hub appears as a single port hub.

As a part of the root hub interface, the PIC32MX USB Driver requires the application to supply functions for hub features that it does not implement. These features are:

- Port Overcurrent Detect
- VBUS Switch Control
- Port Indication

A pointer to these functions (if implemented) must be supplied through the driver initialization data (of the type DRV_USBFS_INIT) structure at the time of driver initialization. The application has the option of not implementing these functions. In such a case, the function pointers for the unimplemented function, in the initialization data structure should be set to NULL.

The root hub driver must also be able to communicate the maximum current capability of its port to the USB Host Layer. The PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for controlling the root hub port current. To facilitate this request, the driver will report the current capability that was specified in the `rootHubAvailableCurrent` parameter of the driver initialization data structure. The application must set this parameter to report the current supply capability of the VBUS power supply. The USB Host Layer uses this value to manage the bus current budget. If a connected device reports a configuration that requires more current than what the VBUS power supply can provide, the host will not set the configuration.

Port Overcurrent Detect

The Root Hub operation in PIC32MX USB Driver will periodically call a Port Overcurrent Detect function to detect if an overcurrent condition is active on the port. The application must supply this function if port overcurrent detection is needed. The PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for checking overcurrent condition. The overcurrent condition on the port can occur in a case where the attached device has malfunctioned or when the USB VBUS line has short circuited to ground.

The signature of the function and an example implementation is shown in the following code example. The function must return (and must continue to return) true if an overcurrent condition exists on the port.

```
/* This code shows an example implementation of the
 * portOverCurrentDetect function. The PIC32MX USB Driver will call this
 * function periodically to check if an over current condition exists on the
 * port. In this example, we assume that the over current detect pin from an
 * external circuit in the system, is connected to  port RD0 and the pin logic
 * is active high. The function must return true if an over current condition is
 * present on this pin */
```

```
bool BSP_USBVBUSSwitchOverCurrentDetect(uint8_t port)
{
    if(PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_D, 0) == 1)
    {
        return(true);
    }
    else
    {
        return(false);
    }
}
```

VBUS Switch Control

The PIC32MX USB Driver Root Hub operation will attempt to control the VBUS power supply to the port. Because the PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for checking controlling VBUS, such a control function must be supplied by the application. The root hub operation will access this function when the PIC32MX USB Driver will call the portPowerEnable function as a part of the Bus Enable sequence.

The following code shows an example of how this function can be implemented.

```
/* This code shows an example implementation of the VBUS Power Enable
 * function. The PIC32MX USB Driver will call this function as a part of bus
 * enable function. In this example, it is assumed that system contains an
 * external VBUS power switch and this is control by port RB5.
 */

void BSP_USBVBUSPowerEnable(uint8_t port, bool enable)
{
    if(enable)
    {
        PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
    else
    {
        PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
}
```

Port Indication function

The Root Hub Operation in the PIC32MX USB Driver allows display of Port LED status. If the application requires this indication, it must implement a function which the Root Hub operation would call when a change in the Root Hub port has occurred. The port indication operation is specified in Section 11.5.3 of the USB 2.0 Specification.

```
/* This code shows an example implementation of the port indication
 * function. The PIC32MX USB Driver calls this function when it wants to indicate
 * port status. It is assumed that three function to switch off, blink and
 * switch on an LED are available. It is further assumed that these function
 * accept the color of the LED to operated on. */

void BSP_RootHubPortIndication
(
    uint8_t port,
    USB_HUB_PORT_INDICATOR_COLOR color,
    USB_HUB_PORT_INDICATOR_STATE state
)
{
    /* The color parameter indicates the color of the LED to be affected. The
     * color will be either USB_HUB_PORT_INDICATOR_COLOR_GREEN or
     * USB_HUB_PORT_INDICATOR_COLOR_AMBER. */

    switch (state)
    {
        case USB_HUB_PORT_INDICATOR_STATE_OFF:
            BSP_SwitchLEDOff(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_BLINKING:
            BSP_LEDBlink(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_ON:
            BSP_SwitchLEDOn(color);
            break;
```

```
            default:
                break;
        }
    }
```

Configuring the Library

Provides information on the configuring the library.

Macros

	Name	Description
	DRV_USBFS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBFS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.
	DRV_USBFS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBFS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBFS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBFS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBFS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBFS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

The PIC32MX USB Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_USBFS_DEVICE_SUPPORT Macro

Determines if the USB Device Functionality should be enabled.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_DEVICE_SUPPORT true
```

Description

USB Full Speed Driver Device Mode Support.

This constant should be set to true if USB device support is required in the application. It should be set to false if device support is not required.

Remarks

This constant should always be defined.

DRV_USBFS_ENDPOINTS_NUMBER Macro

Configures the number of endpoints to be provisioned in the driver.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_ENDPOINTS_NUMBER 3
```

Description

USB Full Speed Driver Endpoint Numbers.

This constant configures the number of endpoints that the driver needs to manage. When DRV_USBFS_DEVICE_SUPPORT is enabled, this constant should be set to the total number of endpoints to be enabled in the device. When enabled, a endpoint can be used for communication.

Using any direction of an endpoint will require that entire endpoint to be enabled.

Consider the case of a composite USB Device that containing a CDC and MSD function. The CDC function will require 1 Bulk endpoint (OUT and IN directions) and 1 Interrupt endpoint (IN direction). The MSD function will require 1 Bulk endpoint (IN and OUT directions). This design can be implemented by using 4 endpoints. Endpoint 0 is used for the mandatory control interface. Endpoint 1 is used for CDC Bulk interface. Endpoint 2 is used for CDC interrupt interface and endpoint 3 is used for MSD Bulk Interface. The constant should then be set to 4.

For Host mode operation, this constant should be set to 1. Setting this to greater than 1 will result in unused data memory allocation.

Remarks

This constant should always be defined.

DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION Macro

Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION 500
```

Description

USB Full Speed Driver Host Mode Attach Debounce Duration.

This constant configures the time duration (in milliseconds) that driver will wait to re-confirm a device attach. When the driver first detects device attach, it start, it will start a timer for the duration specified by the constant. When the timer expires, the driver will check if the device is still attached. If so, the driver will then signal attach to the host stack. The duration allows for device attach to become electro-mechanically stable.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_NAK_LIMIT Macro

Configures the NAK Limit for Host Mode Control Transfers.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_NAK_LIMIT 2000
```

Description

USB Full Speed Driver Host Mode Control Transfers NAK Limit.

This constant configures the number of NAKs that the driver can accept from the device in the data stage of a control transfer before aborting the control transfer with a USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT. Setting this constant to 0 will disable NAK limit checking. This constant should be adjusted to enable USB host compatibility with USB Devices which require more time to process control transfers.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_PIPES_NUMBER Macro

Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_PIPES_NUMBER 10
```

Description

USB Full Speed Driver Host Mode Pipes Number.

This constant configures the maximum number of pipes that can be opened when the driver is operating in Host mode. Calling the DRV_USBFS_HOST_PipeSetup function will cause a pipe to be opened. Calling this function when DRV_USBFS_HOST_PIPES_NUMBER

number of pipes have already been opened will cause the function to return an Invalid Pipe Handle. This constant should be configured to account for the maximum number of devices and the device types to be supported by the host application.

For example if the USB Host application must support 2 USB Mass Storage devices and 1 CDC device, it must set this constant 9 (4 bulk pipes for 2 Mass Storage devices + 2 bulk pipes and 1 interrupt pipe for 1 CDC device and 2 control pipes for 2 devices). Allocating pipes consumes data memory.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_RESET_DURATION Macro

Configures the time duration (in milliseconds) of the Reset Signal.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_RESET_DURATION 100
```

Description

USB Full Speed Driver Host Mode Reset Duration.

This constant configures the duration of the reset signal. The driver generates reset signal when the USB Host stack requests for root hub port reset. The driver will generate the reset signal for the duration specified by this constant and will then stop generating the reset signal.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_SUPPORT Macro

Determines if the USB Host Functionality should be enabled.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_SUPPORT false
```

Description

USB Full Speed Driver Host Mode Support.

This constant should be set to true if USB Host mode support is required in the application. It should be set to false if host support is not required.

Remarks

This constant should always be defined.

DRV_USBFS_INSTANCES_NUMBER Macro

Specifies the number of driver instances to be enabled in the application.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_INSTANCES_NUMBER 1
```

Description

USB Full Speed Driver Instances Number.

This constant defines the number of driver instances to be enabled in the application. This will be typically be the number of USB controllers to be used in the application. On PIC32MX microcontrollers that have one USB controller, this value will always be 1. On PIC32MX microcontrollers which have 2 USB controllers, this value could 1 or 2, depending on whether 1 or 2 USB segments are required. To conserve data memory, this constant should be set to exactly the number of USB controller that are required in the system.

Remarks

This constant should always be defined.

DRV_USBFS_INTERRUPT_MODE Macro

Configures the driver for interrupt or polling mode operation.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_INTERRUPT_MODE true
```

Description

USB Full Speed Driver Interrupt Mode.

This constant configures the driver for interrupt or polling operation. If this flag is set to true, the driver will operate in interrupt mode. If the flag is set to false, the driver will operate in polled mode. In polled, the driver interrupt state machine gets updated in the SYS_Tasks(). If the driver is configured interrupt mode, the driver interrupt state machine gets updated in the driver interrupt service routine. It is always recommended for the driver to operate in interrupt mode.

Remarks

This constant should always be defined.

Building the Library

This section lists the files that are available in the PIC32MX USB Driver Library.

Description

This section list the files that are available in the `\src` folder of the PIC32MX USB Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usb/usbfs`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_usbfs.h	This file should be included by any `.c` file which accesses the PIC32MX USB Driver API. This one file contains the prototypes for all driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_usbfs.c	This file should always be included in the project when using the PIC3MX USB Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/dynamic/drv_usbfs_device.c	This file should be included in the project if Device mode operation is required.
/src/dynamic/drv_usbfs_host.c	This file should be included in the project if Host mode operation is required.

Module Dependencies

The PIC32MX USB Driver Library depends on the following modules:

- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_USBFS_Status	Provides the current status of the USB Driver module.
	DRV_USBFS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBFS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.

b) Client Core Functions

	Name	Description
	DRV_USBFS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBFS_Close	Closes an opened-instance of the USB Driver.
	DRV_USBFS_Initialize	Initializes the USB Driver.
	DRV_USBFS_Open	Opens the specified USB Driver instance and returns a handle to it.

c) Device Mode Operation Functions

	Name	Description
	DRV_USBFS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBFS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBFS_DEVICE_CurrentSpeedGet	This function returns the USB speed at which the device is operating.
	DRV_USBFS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBFS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBFS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBFS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBFS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBFS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBFS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBFS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.

d) Host Mode Operation Functions

	Name	Description
	DRV_USBFS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBFS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBFS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBFS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBFS_HOST_PipeClose	Closes an open pipe.
	DRV_USBFS_HOST_PipeSetup	Open a pipe with the specified attributes.

e) Root Hub Functions

	Name	Description
	DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBFS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.

	DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBFS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBFS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBFS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBFS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.

f) Data Types and Constants

	Name	Description
	DRV_USBFS_EVENT	Identifies the different events that the USB Driver provides.
	DRV_USBFS_EVENT_CALLBACK	Type of the USB Driver event callback function.
	DRV_USBFS_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe Handle type.
	DRV_USBFS_INIT	This type definition defines the Driver Initialization Data Structure.
	DRV_USBFS_OPMODES	Identifies the operating modes supported by the USB Driver.
	DRV_USBFS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).
	DRV_USBFS_DEVICE_INTERFACE	USB Driver Device Mode Interface Functions.
	DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE	USB Driver Endpoint Table Entry Size in bytes.
	DRV_USBFS_HOST_INTERFACE	USB Driver Host Mode Interface Functions.
	DRV_USBFS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBFS_INDEX_0	USB Driver Module Index 0 Definition.
	DRV_USBFS_INDEX_1	USB Driver Module Index 1 Definition.

Description

This section describes the functions of the PIC32MX USB Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_USBFS_Status Function

Provides the current status of the USB Driver module.

File

drv_usbfs.h

C

```
SYS_STATUS DRV_USBFS_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is ready.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has never been initialized.

Description

This function provides the current status of the USB Driver module.

Remarks

None.

Preconditions

The DRV_USBFS_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBFS_Initialize
SYS_STATUS          status;
DRV_USBFS_INIT moduleInit;

uint8_t __attribute__((aligned(512))) endpointTable[DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE * 2];

usbInitData.usbID               = USB_ID_1;
usbInitData.opMode              = DRV_USBFS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);

// The status of the driver can be checked.
status = DRV_USBFS_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USBFS_Initialize function.

Function

SYS_STATUS DRV_USBFS_Status (SYS_MODULE_OBJ object)

DRV_USBFS_Tasks Function

Maintains the driver's state machine when the driver is configured for Polled mode.

File

drv_usbfs.h

C

```
void DRV_USBFS_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Maintains the driver's Polled state machine. This function should be called from the SYS_Tasks function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks). This function will never block.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBFS_Initialize

while (true)
{
    DRV_USBFS_Tasks(object);
```

```
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBFS_Initialize function).

Function

void DRV_USBFS_Tasks(SYS_MODULE_OBJ object)

DRV_USBFS_Tasks_ISR Function

Maintains the driver's Interrupt state machine and implements its ISR.

File

drv_usbfs.h

C

```
void DRV_USBFS_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal Interrupt state machine and implement its ISR for interrupt-driven implementations.

Remarks

This routine should be called from the USB interrupt service routine. In case of multiple USB modules, it should be ensured that the correct USB driver system module object is passed to this routine.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBFS_Initialize

while (true)
{
    DRV_USBFS_Tasks_ISR (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBFS_Initialize).

Function

void DRV_USBFS_Tasks_ISR(SYS_MODULE_OBJ object)

b) Client Core Functions

DRV_USBFS_ClientEventCallBackSet Function

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.

File

drv_usbfs.h

C

```
void DRV_USBFS_ClientEventCallBackSet(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK
```

```
myEventCallBack);
```

Returns

None.

Description

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events. The callback is disabled by either not calling this function after the DRV_USBFS_Open function has been called or by setting the myEventCallBack argument as NULL. When the callback function is called, the hReferenceData argument is returned.

Remarks

Typical usage of the USB Driver requires a client to register a callback.

Preconditions

None.

Example

```
// Set the client event callback for the Device Layer.  The
// USBDeviceLayerEventHandler function is the event handler. When this
// event handler is invoked by the driver, the driver returns back the
// second argument specified in the following function (which in this case
// is the Device Layer data structure). This allows the application
// firmware to identify, as an example, the Device Layer object associated
// with this callback.

DRV_USBFS_ClientEventCallBackSet(myUSBDevice.usbDriverHandle, (uintptr_t)&myUSBDevice,
USBDeviceLayerEventHandler);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
hReferenceData	Object (could be a pointer) that is returned with the callback.
myEventCallBack	Callback function for all USB events.

Function

void DRV_USBFS_ClientEventCallBackSet

(

DRV_HANDLE handle,

uintptr_t hReferenceData,

DRV_USBFS_EVENT_CALLBACK myEventCallBack

);

DRV_USBFS_Close Function

Closes an opened-instance of the USB Driver.

File

drv_usbfs.h

C

```
void DRV_USBFS_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the USB Driver, invalidating the handle.

Remarks

After calling this function, the handle passed in handle parameter must not be used with any of the other driver functions. A new handle must be obtained by calling DRV_USBFS_Open function before the caller may use the driver again.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance. DRV_USBFS_Open function must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USBFS_Open

DRV_USBFS_Close(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_Close(DRV_HANDLE handle)

DRV_USBFS_Initialize Function

Initializes the USB Driver.

File

drv_usbfs.h

C

```
SYS_MODULE_OBJ DRV_USBFS_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

- SYS_MODULE_OBJ_INVALID - The driver initialization failed.
- A valid System Module Object - The driver initialization was able to start. It may have not completed and requires the DRV_USBFS_Tasks function to be called periodically. This value will never be the same as SYS_MODULE_OBJ_INVALID.

Description

This function initializes the USB Driver, making it ready for clients to open. The driver initialization does not complete when this function returns. The DRV_USBFS_Tasks function must called periodically to complete the driver initialization. The DRV_USBHS_Open function will fail if the driver was not initialized or if initialization has not completed.

Remarks

This routine must be called before any other USB driver routine is called. This routine should only be called once during system initialization unless DRV_USBFS_Deinitialize is called to deinitialize the driver instance.

Preconditions

None.

Example

```
 // The following code shows an example initialization of the
 // driver. The USB module to be used is USB1.  The module should not
 // automatically suspend when the  microcontroller enters Sleep mode.  The
 // module should continue  operation when the CPU enters Idle mode.  The
 // power state is set to run at full clock speeds. Device Mode operation
 // should be at FULL speed. The size of the endpoint table is set for 2
 // endpoints.

DRV_USBFS_INIT moduleInit;

uint8_t __attribute__((aligned(512))) endpointTable[DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE * 2];

usbInitData.usbID               = USB_ID_1;
usbInitData.opMode              = DRV_USBFS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;
```

```
// This is how this data structure is passed to the initialize
// function.

DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

Parameters

Parameters	Description
drvIndex	Ordinal number of driver instance to be initialized. This should be set to DRV_USBFS_INDEX_0 if driver instance 0 needs to be initialized.
init	Pointer to a data structure containing data necessary to initialize the driver. This should be a DRV_USBFS_INIT structure reference typecast to SYS_MODULE_INIT reference.

Function

SYS_MODULE_OBJ DRV_USBHS_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT * const init

)

DRV_USBFS_Open Function

Opens the specified USB Driver instance and returns a handle to it.

File

drv_usbfs.h

C

```
DRV_HANDLE DRV_USBFS_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE_INVALID - The driver could not be opened successfully.This can happen if the driver initialization was not complete or if an internal error has occurred.
- A Valid Driver Handle - This is an arbitrary value and is returned if the function was successful. This value will never be the same as DRV_HANDLE_INVALID.

Description

This function opens the specified USB Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The intent flag should always be DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NON_BLOCKING. Any other setting of the intent flag will return a invalid driver handle. A driver instance can only support one client. Trying to open a driver that has an existing client will result in an unsuccessful function call.

Remarks

The handle returned is valid until the DRV_USBFS_Close function is called. The function will typically return DRV_HANDLE_INVALID if the driver was not initialized. In such a case the client should try to open the driver again.

Preconditions

Function DRV_USBFS_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

// This code assumes that the driver has been initialized.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

if(DRV_HANDLE_INVALID == handle)
{
    // The application should try opening the driver again.
}
```

Parameters

Parameters	Description
drvIndex	Identifies the driver instance to be opened. As an example, this value can be set to DRV_USBFS_INDEX_0 if instance 0 of the driver has to be opened.
intent	Should always be (DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE| DRV_IO_INTENT_NON_BLOCKING).

Function

```
DRV_HANDLE DRV_USBFS_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent
)
```

c) Device Mode Operation Functions

DRV_USBFS_DEVICE_AddressSet Function

This function will set the USB module address that is obtained from the Host.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);
```

Returns

None.

Description

This function will set the USB module address that is obtained from the Host in a setup transaction. The address is obtained from the SET_ADDRESS command issued by the Host. The primary (first) client of the driver uses this function to set the module's USB address after decoding the setup transaction from the Host.

Remarks

None.

Preconditions

None.

Example

```
// This function should be called by the first client of the driver,
// which is typically the Device Layer. The address to set is obtained
// from the Host during enumeration.

DRV_USBFS_DEVICE_AddressSet(deviceLayer, 4);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
address	The address of this module on the USB bus.

Function

void DRV_USBFS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);

DRV_USBFS_DEVICE_Attach Function

This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on

the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_Attach(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables the pull-up resistors on the D+ or D- lines thus letting the USB Host know that a device has been attached on the bus . This function should be called when the driver client is ready to receive communication from the Host (typically after all initialization is complete). The USB 2.0 specification requires VBUS to be detected before the data line pull-ups are enabled. The application must ensure the same.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBFS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be attached when VBUS Session Valid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBFS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBFS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBFS_DEVICE_Attach(handle);
            break;

        case DRV_USBFS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBFS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_Attach(DRV_HANDLE handle);

DRV_USBFS_DEVICE_CurrentSpeedGet Function

This function returns the USB speed at which the device is operating.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_ERROR - The device speed is not valid.
- USB_SPEED_FULL - The device is operating at Full speed.

Description

This function returns the USB speed at which the device is operating.

Remarks

None.

Preconditions

Only valid after the device is attached to the Host and Host has completed reset signaling.

Example

```
// Get the current speed.

USB_SPEED deviceSpeed;

deviceSpeed = DRV_USBFS_DEVICE_CurrentSpeedGet(deviceLayer);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

USB_SPEED DRV_USBFS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);

DRV_USBFS_DEVICE_Detach Function

This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_Detach(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables the pull-up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to Host mode operation). A self-powered device should be detached from the bus when the VBUS is not valid.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBFS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);
```

```
// The device can be detached when VBUS Session Invalid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBFS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBFS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBFS_DEVICE_Attach(handle);
            break;

        case DRV_USBFS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBFS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_Detach(DRV_HANDLE handle);

DRV_USBFS_DEVICE_EndpointDisable Function

This function disables an endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointDisable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The endpoint that is being accessed is not a valid endpoint (endpoint was not provisioned through the DRV_USBFS_ENDPOINTS_NUMBER configuration constant) defined for this driver instance.

Description

This function disables an endpoint. If the endpoint type is a control endpoint type, both directions are disabled. For non-control endpoints, the function disables the specified direction only. The direction to be disabled is specified by the Most Significant Bit (MSB) of the endpointAndDirection parameter.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable
// a control endpoint. Note that the direction parameter is ignored.
// For a control endpoint, both the directions are disabled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBFS_DEVICE_EndpointDisable(handle, ep );
```

```
// This code shows an example of how to disable a BULK IN
// endpoint

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointDisable(handle, ep );
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointDisable
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBFS_DEVICE_EndpointDisableAll Function

This function disables all provisioned endpoints.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointDisableAll(DRV_HANDLE handle);
```

Returns

- USB_ERROR_NONE - The function exited successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function disables all provisioned endpoints in both directions.

Remarks

This function is typically called by the USB Device Layer to disable all endpoints upon detecting a bus reset.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable all endpoints.

DRV_USBFS_DEVICE_EndpointDisableAll(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointDisableAll(DRV_HANDLE handle)

DRV_USBFS_DEVICE_EndpointEnable Function

This function enables an endpoint for the specified direction and endpoint size.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointEnable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is not a valid endpoint defined for this driver instance. The value of DRV_USBFS_ENDPOINTS_NUMBER configuration constant should be adjusted.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function enables an endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, with the exception of control endpoints. If the endpoint type is a control endpoint, the endpoint is always bidirectional and the function needs to be called only once.

The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. A transfer that is scheduled over this endpoint will be scheduled in wMaxPacketSize transactions. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally reused.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to enable Endpoint
// 0 for control transfers. Note that for a control endpoint, the
// direction parameter is ignored. A control endpoint is always
// bidirectional. Endpoint size is 64 bytes.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);

// This code shows an example of how to set up a endpoint
// for BULK IN transfer. For an IN transfer, data moves from device
// to Host. In this example, Endpoint 1 is enabled. The maximum
// packet size is 64.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);

// If Endpoint 1 must also be set up for BULK OUT, the
// DRV_USBFS_DEVICE_EndpointEnable function must be called again, as shown
// here.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Function

```
USB_ERROR DRV_USBFS_DEVICE_EndpointEnable
(
    DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType,
uint16_t endpointSize
);
```

DRV_USBFS_DEVICE_EndpointIsEnabled Function

This function returns the enable/disable status of the specified endpoint and direction.

File

drv_usbfs.h

C

```
bool DRV_USBFS_DEVICE_EndpointIsEnabled(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- true - The endpoint is enabled.
- false - The endpoint is disabled.

Description

This function returns the enable/disable status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBFS_DEVICE_EndpointIsEnabled function can be used to obtain the
// status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(DRV_USBFS_ENDPOINT_STATE_DISABLED ==
            DRV_USBFS_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBFS_DEVICE_EndpointIsEnabled
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBFS_DEVICE_EndpointIsStalled Function

This function returns the stall status of the specified endpoint and direction.

File

drv_usbfs.h

C

```
bool DRV_USBFS_DEVICE_EndpointIsStalled(DRV_HANDLE client, USB_ENDPOINT endpoint);
```

Returns

- true - The endpoint is stalled.
- false - The endpoint is not stalled.

Description

This function returns the stall status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBFS_DEVICE_EndpointIsStalled function can be used to obtain the
// stall status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(true == DRV_USBFS_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    DRV_USBFS_DEVICE_EndpointStallClear(handle, ep);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBFS_DEVICE_EndpointIsStalled
(

DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBFS_DEVICE_EndpointStall Function

This function stalls an endpoint in the specified direction.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointStall(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_OSAL_FUNCTION - An error with an OSAL function called in this function.

Description

This function stalls an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to stall an endpoint. In
// this example, Endpoint 1 IN direction is stalled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointStall(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointStall
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBFS_DEVICE_EndpointStallClear Function

This function clears the stall on an endpoint in the specified direction.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointStallClear(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Description

This function clears the stall on an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to clear a stall. In this
// example, the stall condition on Endpoint 1 IN direction is cleared.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointStallClear(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointStallClear

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBFS_DEVICE_IRPCancel Function

This function cancels the specific IRP that are queued and in progress at the specified endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancel(DRV_HANDLE client, USB_DEVICE_IRP * irp);
```

Returns

- USB_ERROR_NONE - The IRP have been canceled successfully.
- USB_ERROR_PARAMETER_INVALID - Invalid parameter or the IRP already has been aborted or completed
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function attempts to cancel the processing of a queued IRP. An IRP that was in the queue but yet to be processed will be cancelled successfully and the IRP callback function will be called from this function with the USB_DEVICE_IRP_STATUS_ABORTED status. The application can release the data buffer memory used by the IRP when this callback occurs. If the IRP was in progress (a transaction in on the bus) when the cancel function was called, the IRP will be canceled only when an ongoing or the next transaction has completed. The IRP callback

function will then be called in an interrupt context. The application should not release the related data buffer unless the IRP callback has occurred.

Remarks

The size returned after the ABORT callback will be always 0 regardless of the amount of data that has been sent or received. The client should not assume any data transaction has happened for an canceled IRP. If the last transaction of the IRP was in progress, the IRP cancel does not have any effect. The first transaction of any ongoing IRP cannot be canceled.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel IRP.  In this example the IRP
// has been scheduled from a device to the Host.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBFS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // Check the status of the IRP.
    if(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Cancel the submitted IRP.
        if (DRV_USBFS_DEVICE_IRPCancel(handle, &irp) != USB_ERROR_NONE)
        {
            // The IRP Cancel request submission was successful.
            // IRP cancel status will be notified through the callback
            // function.
        }
        else
        {
            // The IRP may have been completed before IRP cancel operation.
            // could start. No callback notification will be generated.
        }
    }
    else
    {
        // The IRP processing must have been completed before IRP cancel was
        // submitted.
    }
}

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if the IRP callback is for a Cancel request
    if(irp->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        // IRP cancel completed
    }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
irp	Pointer to the IRP to cancel.

Function

USB_ERROR DRV_USBFS_DEVICE_IRPCancel
(
DRV_HANDLE client,
USB_DEVICE_IRP * irp
)

DRV_USBFS_DEVICE_IRPCancelAll Function

This function cancels all IRPs that are queued and in progress at the specified endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancelAll(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel all IRPs.

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if this is setup command

    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)
    {
        if(IsSetupCommandSupported(irp->data) == false)
        {
            // This means that this setup command is not
            // supported. Stall the some related endpoint and cancel all
            // queue IRPs.

            DRV_USBFS_DEVICE_EndpointStall(handle, ep);
            DRV_USBFS_DEVICE_IRPCancelAll(handle, ep);
        }
     }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBFS_DEVICE_IRPCancelAll

```
(
    DRV_HANDLE client,
USB_ENDPOINT endpointAndDirection
);
```

DRV_USBFS_DEVICE_IRPSubmit Function

This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPSubmit(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection, USB_DEVICE_IRP *
irp);
```

Returns

- USB_ERROR_NONE - if the IRP was submitted successful.
- USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct.
- USB_ERROR_PARAMETER_INVALID - If the client handle is not valid.
- USB_ERROR_ENDPOINT_NOT_CONFIGURED - If the endpoint is not enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The specified endpoint is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL call in the function did not complete successfully.

Description

This function submits an I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to send and receive data from the USB Host. The data will be sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection parameter. Submitting an IRP arms the endpoint to either send data to or receive data from the Host. If an IRP is already being processed on the endpoint, the subsequent IRP submit operation will be queued. The contents of the IRP (including the application buffers) should not be changed until the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the Host can be less than, greater than, equal to, or be an exact multiple of the maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter, while sending data to the Host is less than the maximum packet size, the transfer will complete in one transaction.
- If the size parameter, while sending data to the Host is greater than the maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter, while sending data to the Host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to send a Zero Length Packet(ZLP) by specifying the USB_DEVICE_IRP_FLAG_DATA_COMPLETE flag as the flag parameter.
- The size parameter, while receiving data from the Host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_ERROR_IRP_SIZE_INVALID result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the Host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the Host. The IRP status is returned as USB_DEVICE_IRP_STATUS_COMPLETED_SHORT.
- If a ZLP needs to be sent to Host, the IRP size should be specified as 0 and the flag parameter should be set as USB_DEVICE_IRP_FLAG_DATA_COMPLETE.
- If the IRP size is an exact multiple of the endpoint size, the client can request the driver to not send a ZLP by setting the flag parameter to USB_DEVICE_IRP_FLAG_DATA_PENDING. This flag indicates that there is more data pending in this transfer.
- Specifying a size less than the endpoint size along with the USB_DEVICE_IRP_FLAG_DATA_PENDING flag will cause the driver to return a USB_ERROR_IRP_SIZE_INVALID.
- If the size is greater than but not a multiple of the endpoint size, and the flag is specified as USB_DEVICE_IRP_FLAG_DATA_PENDING, the driver will send multiple of endpoint size number of bytes. For example, if the IRP size is 130 and the endpoint size if 64, the number of bytes sent will 128.

Remarks

This function can be called from the ISR of the USB module to associated with the client.

Preconditions

The Client handle should be valid.

Example

```
// The following code shows an example of how to schedule a IRP to send data
// from a device to the Host. Assume that the max packet size is 64 and
// and this data needs to sent over Endpoint 1. In this example, the
```

```
// transfer is processed as three transactions of 64, 64 and 2 bytes.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBFS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task function.
    }
}

// The following code shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
// endpoint size.

irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

// Note that while  receiving data from the Host, the size should be an
// exact multiple of the maximum packet size of the endpoint. In the
// following example, the DRV_USBFS_DEVICE_IRPSubmit function will return a
// USB_DEVICE_IRP_SIZE_INVALID value.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

irp.data = myDataBufferToSend;
irp.size = 60; // THIS SIZE IS NOT CORRECT
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the IRP to be added to the queue for processing.

Function

USB_ERROR DRV_USBFS_DEVICE_IRPSubmit

(

DRV_HANDLE client,

USB_ENDPOINT endpointAndDirection,

USB_DEVICE_IRP * irp

);

DRV_USBFS_DEVICE_RemoteWakeupStart Function

This function causes the device to start Remote Wakeup Signalling on the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to start Remote Wakeup Signalling on the bus. This function should be called when the device, presently placed in suspend mode by the Host, wants to be wakeup. Note that the device can do this only when the Host has enabled the device's Remote Wakeup capability.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling.

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBFS_DEVICE_RemoteWakeupStart(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);

DRV_USBFS_DEVICE_RemoteWakeupStop Function

This function causes the device to stop the Remote Wakeup Signalling on the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to stop Remote Wakeup Signalling on the bus. This function should be called after the DRV_USBFS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Remarks

This function should be 1 to 15 milliseconds after the DRV_USBFS_DEVICE_RemoteWakeupStart function was called.

Preconditions

The handle should be valid. The DRV_USBFS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling. Wait for 10
// milliseconds and then stop the Remote Wakeup signaling

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBFS_DEVICE_RemoteWakeupStart(handle);
    DelayMilliSeconds(10);
    DRV_USBFS_DEVICE_RemoteWakeupStop(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);

DRV_USBFS_DEVICE_SOFNumberGet Function

This function will return the USB SOF packet number.

File

drv_usbfs.h

C

```
uint16_t DRV_USBFS_DEVICE_SOFNumberGet(DRV_HANDLE handle);
```

Returns

The SOF packet number.

Description

This function will return the USB SOF packet number..

Remarks

None.

Preconditions

This function will return a valid value only when the device is attached to the bus. The SOF packet count will not increment if the bus is suspended.

Example

```
// This code shows how the DRV_USBFS_DEVICE_SOFNumberGet function is called
// to read the current SOF number.

DRV_HANDLE handle;
uint16_t sofNumber;

sofNumber = DRV_USBFS_DEVICE_SOFNumberGet(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

uint16_t DRV_USBFS_DEVICE_SOFNumberGet(DRV_HANDLE handle);

d) Host Mode Operation Functions

DRV_USBFS_HOST_EventsDisable Function

Disables Host mode events.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_EventsDisable(DRV_HANDLE handle);
```

Returns

- true - Driver event generation was enabled when this function was called.
- false - Driver event generation was not enabled when this function was called.

Description

This function disables the Host mode events. This function is called by the Host Layer when it wants to execute code atomically.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_EventsDisable and
// DRV_USBFS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBFS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBFS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

bool DRV_USBFS_HOST_EventsDisable

(

DRV_HANDLE handle

);

DRV_USBFS_HOST_EventsEnable Function

Restores the events to the specified the original value.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_EventsEnable(DRV_HANDLE handle, bool eventContext);
```

Returns

None.

Description

This function will restore the enable disable state of the events. The eventRestoreContext parameter should be equal to the value returned by the DRV_USBFS_HOST_EventsDisable function.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_EventsDisable and
// DRV_USBFS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBFS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBFS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
eventRestoreContext	Value returned by the DRV_USBFS_HOST_EventsDisable function.

Function

void DRV_USBFS_HOST_EventsEnable

(

DRV_HANDLE handle

bool eventRestoreContext

);

DRV_USBFS_HOST_IRPCancel Function

Cancels the specified IRP.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);
```

Returns

None.

Description

This function attempts to cancel the specified IRP. If the IRP is queued and its processing has not started, it will be cancelled successfully. If the IRP in progress, the ongoing transaction will be allowed to complete.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how a submitted IRP can be cancelled.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);

// Additional application logic may come here. This logic may decide to
// cancel the submitted IRP.

DRV_USBFS_HOST_IRPCancel(&irp);
```

Parameters

Parameters	Description
inputIRP	Pointer to the IRP to cancel.

Function

void DRV_USBFS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);

DRV_USBFS_HOST_IRPSubmit Function

Submits an IRP on a pipe.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_IRPSubmit(DRV_USBFS_HOST_PIPE_HANDLE hPipe, USB_HOST_IRP * pinputIRP);
```

Returns

- USB_ERROR_NONE - The IRP was submitted successfully.
- USB_ERROR_PARAMETER_INVALID - The pipe handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An error occurred in an OSAL function called in this function.

Description

This function submits an IRP on the specified pipe. The IRP will be added to the queue and will be processed in turn. The data will be transferred on the bus based on the USB bus scheduling rules. When the IRP has been processed, the callback function specified in the IRP will be called. The IRP status will be updated to reflect the completion status of the IRP.

Remarks

An IRP can also be submitted in an IRP callback function.

Preconditions

The pipe handle should be valid.

Example

```
// The following code shows an example of how the host layer populates
// the IRP object and then submits it. IRP_Callback function is called when an
// IRP has completed processing. The status of the IRP at completion can be
// checked in the status flag. The size field of the irp will contain the amount
// of data transferred.

void IRP_Callback(USB_HOST_IRP * irp)
{
    // irp is pointing to the IRP for which the callback has occurred. In most
    // cases this function will execute in an interrupt context. The application
    // should not perform any hardware access or interrupt un-safe operations in
    // this function.

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            // IRP was terminated due to an unknown error
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            // IRP was terminated by the application
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            // IRP was terminated due to a bus error
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            // IRP was terminated due to data error
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            // IRP was terminated because of a NAK timeout
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            // IRP was terminated because of a device sent a STALL
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            // IRP has been completed
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            // IRP has been completed but the amount of data processed was less
            // than requested.
            break;

        default:
            break;
    }
}

// In the following code snippet the a control transfer IRP is submitted to a
// control pipe. The setup parameter of the IRP points to the Setup command of
// the control transfer. The direction of the data stage is specified by the
// Setup packet.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];
```

```
irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);
```

Parameters

Parameters	Description
hPipe	Handle to the pipe to which the IRP has to be submitted.
pInputIRP	Pointer to the IRP.

Function

USB_ERROR DRV_USBFS_HOST_IRPSubmit

(

DRV_USBFS_HOST_PIPE_HANDLE hPipe,

USB_HOST_IRP * pInputIRP

);

DRV_USBFS_HOST_PipeClose Function

Closes an open pipe.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_PipeClose(DRV_USBFS_HOST_PIPE_HANDLE pipeHandle);
```

Returns

None.

Description

This function closes an open pipe. Any IRPs scheduled on the pipe will be aborted and IRP callback functions will be called with the status as DRV_USB_HOST_IRP_STATE_ABORTED. The pipe handle will become invalid and the pipe will not accept IRPs.

Remarks

None.

Preconditions

The pipe handle should be valid.

Example

```
// This code shows how an open Host pipe can be closed.

DRV_HANDLE driverHandle;
DRV_USBFS_HOST_PIPE_HANDLE pipeHandle;

// Close the pipe.
DRV_USBFS_HOST_PipeClose(pipeHandle);
```

Parameters

Parameters	Description
pipeHandle	Handle to the pipe to close.

Function

void DRV_USBFS_HOST_PipeClose

(

```
    DRV_USBFS_HOST_PIPE_HANDLE pipeHandle
);
```

DRV_USBFS_HOST_PipeSetup Function

Open a pipe with the specified attributes.

File

drv_usbfs.h

C

```
DRV_USBFS_HOST_PIPE_HANDLE DRV_USBFS_HOST_PipeSetup(DRV_HANDLE client, uint8_t deviceAddress, USB_ENDPOINT
endpointAndDirection, uint8_t hubAddress, uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval,
uint16_t wMaxPacketSize, USB_SPEED speed);
```

Returns

- DRV_USB_HOST_PIPE_HANDLE_INVALID - The pipe could not be created.
- A valid Pipe Handle - The pipe was created successfully. This is an arbitrary value and will never be the same as DRV_USB_HOST_PIPE_HANDLE_INVALID.

Description

This function opens a communication pipe between the Host and the device endpoint. The transfer type and other attributes are specified through the function parameters. The driver does not check for available bus bandwidth, which should be done by the application (the USB Host Layer in this case)

Remarks

None.

Preconditions

The driver handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_PipeSetup function is called for
// create a communication pipe. In this example, Bulk pipe is created
// between the Host and a device. The Device address is 2 and the target
// endpoint on this device is 4 . The direction of the data transfer over
// this pipe is from the Host to the device. The device is connected to Port
// 1 of a Hub, whose USB address is 3. The maximum size of a transaction
// on this pipe is 64 bytes. This is a Bulk Pipe and hence the bInterval
// field is set to 0. The target device is operating at Full Speed.

DRV_HANDLE driverHandle;
DRV_USBFS_HOST_PIPE_HANDLE pipeHandle;

pipeHandle = DRV_USBFS_HOST_PipeSetup(driverHandle, 0x02, 0x14, 0x03, 0x01, USB_TRANSFER_TYPE_BULK, 0, 64,
USB_SPEED_FULL);

if(pipeHandle != DRV_USBFS_HOST_PIPE_HANDLE_INVALID)
{
    // The pipe was created successfully.
}
```

Parameters

Parameters	Description
client	Handle to the driver (returned from DRV_USBFS_Open function).
deviceAddress	USB Address of the device to connect to.
endpoint	Endpoint on the device to connect to.
hubAddress	Address of the hub to which this device is connected. If not connected to a hub, this value should be set to 0.
hubPort	Port number of the hub to which this device is connected.
pipeType	Transfer type of the pipe to open.
bInterval	Polling interval for periodic transfers. This should be specified as defined by the USB 2.0 Specification.

wMaxPacketSize	This should be set to the endpoint size reported by the device in its configuration descriptors. This defines the maximum size of the transaction in a transfer on this pipe.
speed	The speed of the pipe. This should match the speed at which the device connected to the Host.

Function

```
DRV_USBFS_HOST_PIPE_HANDLE DRV_USBFS_HOST_PipeSetup
(
    DRV_HANDLE client,
uint8_t deviceAddress,
USB_ENDPOINT endpointAndDirection,
uint8_t hubAddress,
uint8_t hubPort,
USB_TRANSFER_TYPE pipeType,
uint8_t bInterval,
uint16_t wMaxPacketSize,
USB_SPEED speed
);
```

e) Root Hub Functions

DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet Function

This function returns the operating speed of the bus to which this root hub is connected.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_FULL - The Root hub is connected to a bus that is operating at Full Speed.

Description

This function returns the operating speed of the bus to which this root hub is connected.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet function is
// called to know the operating speed of the bus to which this Root hub is
// connected.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

USB_SPEED DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_Initialize Function

This function initializes the root hub driver.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_ROOT_HUB_Initialize(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

Returns

None.

Description

This function initializes the root hub driver. It is called by the Host Layer at the time of processing the root hub devices. The Host Layer assigns a USB_HOST_DEVICE_INFO reference to this root hub driver. This identifies the relationship between the root hub and the Host Layer.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the USB Host Layer calls the
// DRV_USBFS_HOST_ROOT_HUB_Initialize function. The usbHostDeviceInfo
// parameter is an arbitrary identifier assigned by the USB Host Layer. Its
// interpretation is opaque to the Root hub Driver.

DRV_HANDLE drvHandle;
USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo = 0x10003000;

DRV_USBFS_HOST_ROOT_HUB_Initialize(drvHandle, usbHostDeviceInfo);
```

Parameters

Parameters	Description
handle	Handle to the driver.
usbHostDeviceInfo	Reference provided by the Host.

Function

void DRV_USBFS_HOST_ROOT_HUB_Initialize

(

DRV_HANDLE handle,

USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo,

)

DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet Function

Returns the maximum amount of current that this root hub can provide on the bus.

File

drv_usbfs.h

C

```
uint32_t DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE handle);
```

Returns

Returns the maximum current (in milliamperes) that the root hub can supply.

Description

This function returns the maximum amount of current that this root hub can provide on the bus.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet
// function is called to obtain the maximum VBUS current that the Root hub
// can supply.

DRV_HANDLE driverHandle;
uint32_t currentMilliAmperes;

currentMilliAmperes = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

uint32_t DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE);

DRV_USBFS_HOST_ROOT_HUB_OperationEnable Function

This function enables or disables root hub operation.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_ROOT_HUB_OperationEnable(DRV_HANDLE handle, bool enable);
```

Returns

None.

Description

This function enables or disables root hub operation. When enabled, the root hub will detect devices attached to the port and will request the Host Layer to enumerate the device. This function is called by the Host Layer when it is ready to receive enumeration requests from the Host. If the operation is disabled, the root hub will not detect attached devices.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
```

```
DRV_USBFS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBFS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
enable	If this is set to true, root hub operation is enabled. If this is set to false, root hub operation is disabled.

Function

void DRV_USBFS_HOST_ROOT_HUB_OperationEnable

(

DRV_HANDLE handle,

bool enable

);

DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled Function

Returns the operation enabled status of the root hub.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);
```

Returns

- true - Root hub operation is enabled.
- false - Root hub operation is not enabled.

Description

This function returns true if the DRV_USBFS_HOST_ROOT_HUB_OperationEnable function has completed enabling the Host.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBFS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBFS_Tasks function
```

```
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

bool DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet Function

Returns the number of ports this root hub contains.

File

drv_usbfs.h

C

```
uint8_t DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);
```

Returns

This function will always return 1.

Description

This function returns the number of ports that this root hub contains.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet function can
// be called to obtain the number of Root hub ports.

DRV_HANDLE driverHandle;
uint8_t nPorts;

nPorts = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

uint8_t DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_PortReset Function

Resets the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);
```

Returns

None.

Description

This function resets the root hub port. The reset duration is defined by DRV_USBFS_ROOT_HUB_RESET_DURATION. The status of the reset signaling can be checked using the DRV_USBFS_ROOT_HUB_PortResetIsComplete function.

Remarks

The root hub on the PIC32MZ USB controller contains only one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBFS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBFS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBFS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to reset.

Function

void DRV_USBFS_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);

DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete Function

Returns true if the root hub has completed the port reset operation.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete(DRV_HANDLE handle, uint8_t port);
```

Returns

- true - The reset signaling has completed.
- false - The reset signaling has not completed.

Description

This function returns true if the port reset operation has completed. It should be called after the DRV_USB_HOST_ROOT_HUB_PortReset function to check if the reset operation has completed.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBFS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBFS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBFS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to check

Function

bool DRV_USBFS_ROOT_HUB_PortResetIsComplete

(

DRV_HANDLE handle,

uint8_t port

);

DRV_USBFS_HOST_ROOT_HUB_PortResume Function

Resumes the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortResume(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function resumes the root hub. The resume duration is defined by DRV_USBFS_ROOT_HUB_RESUME_DURATION. The status of the resume signaling can be checked using the DRV_USBFS_ROOT_HUB_PortResumeIsComplete function.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortResume function is
// called to resume the specified port.

DRV_HANDLE driverHandle;

// Resume Port 0.
DRV_USBFS_HOST_ROOT_HUB_PortResume(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to resume.

Function

USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortResume
(
DRV_HANDLE handle,
uint8_t port
);

DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet Function

Returns the speed of at which the port is operating.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_SPEED_ERROR - This value is returned if the driver handle is not or if the speed information is not available or if the specified port is not valid.
- USB_SPEED_FULL - A Full Speed device has been connected to the port.
- USB_SPEED_LOW - A Low Speed device has been connected to the port.

Description

This function returns the speed at which the port is operating.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet function is
// called to know the operating speed of the port. This also indicates the
// operating speed of the device connected to this port.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port number of the port to be analyzed..

Function

USB_SPEED DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet
(
DRV_HANDLE handle,
uint8_t port
);

DRV_USBFS_HOST_ROOT_HUB_PortSuspend Function

Suspends the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function suspends the root hub port.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortSuspend function is
// called to suspend the specified port.

DRV_HANDLE driverHandle;

// Suspend Port 0.
DRV_USBFS_HOST_ROOT_HUB_PortSuspend(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to suspend.

Function

USB_ERROR DRV_USBFS_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);

f) Data Types and Constants

DRV_USBFS_EVENT Enumeration

Identifies the different events that the USB Driver provides.

File

drv_usbfs.h

C

```
typedef enum {
  DRV_USBFS_EVENT_ERROR = DRV_USB_EVENT_ERROR,
  DRV_USBFS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT,
  DRV_USBFS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT,
  DRV_USBFS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT,
  DRV_USBFS_EVENT_STALL = DRV_USB_EVENT_STALL,
  DRV_USBFS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT,
  DRV_USBFS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID,
  DRV_USBFS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID
} DRV_USBFS_EVENT;
```

Members

Members	Description
DRV_USBFS_EVENT_ERROR = DRV_USB_EVENT_ERROR	Bus error occurred and was reported
DRV_USBFS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT	Host has issued a device reset
DRV_USBFS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT	Resume detected while USB in suspend mode
DRV_USBFS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT	Idle detected
DRV_USBFS_EVENT_STALL = DRV_USB_EVENT_STALL	Stall handshake has occurred
DRV_USBFS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT	Either Device received SOF or SOF threshold was reached in the Host mode operation
DRV_USBFS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID	Session valid
DRV_USBFS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID	Session Invalid

Description

USB Driver Events Enumeration.

This enumeration identifies the different events that are generated by the USB Driver.

Remarks

None.

DRV_USBFS_EVENT_CALLBACK Type

Type of the USB Driver event callback function.

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_EVENT_CALLBACK)(uintptr_t hClient, DRV_USBFS_EVENT eventType, void * eventData);
```

Returns

None.

Description

Type of the USB Driver Event Callback Function.

Define the type of the USB Driver event callback function. The client should register an event callback function of this type when it intends to receive events from the USB Driver. The event callback function is registered using the DRV_USBFS_ClientEventCallBackSet function.

Remarks

None.

Parameters

Parameters	Description
hClient	Handle to the driver client that registered this callback function.
eventType	This parameter identifies the event that caused the callback function to be called.
eventData	Pointer to a data structure that is related to this event. This value will be NULL if the event has no related data.

DRV_USBFS_HOST_PIPE_HANDLE Type

Defines the USB Driver Host Pipe Handle type.

File

drv_usbfs.h

C

```
typedef uintptr_t DRV_USBFS_HOST_PIPE_HANDLE;
```

Description

USB Driver Host Pipe Handle.

This type definition defines the type of the USB Driver Host Pipe Handle.

Remarks

None.

DRV_USBFS_INIT Structure

This type definition defines the Driver Initialization Data Structure.

File

drv_usbfs.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  USB_MODULE_ID usbID;
  bool stopInIdle;
  bool suspendInSleep;
  INT_SOURCE interruptSource;
  USB_SPEED operationSpeed;
  DRV_USBFS_OPMODES operationMode;
  void * endpointTable;
  uint32_t rootHubAvailableCurrent;
  DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;
  DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;
  DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;
} DRV_USBFS_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System Module Initialization
USB_MODULE_ID usbID;	Identifies the USB peripheral to be used. This should be the USB PLIB module instance identifier.
bool stopInIdle;	This should be set to true if the USB module must stop operation in IDLE mode
bool suspendInSleep;	This should be set to true if the USB module must suspend when the CPU enters sleep mode.
INT_SOURCE interruptSource;	Specify the interrupt source for the USB module. This should be the interrupt source identifier for the USB module instance specified in usbID.
USB_SPEED operationSpeed;	Specify the operational speed of the USB module. This should always be set to USB_SPEED_FULL.
DRV_USBFS_OPMODES operationMode;	Specify the operation mode of the USB module. This specifies if the USB module should operate as a Device, Host, or both (Dual Role operation).
void * endpointTable;	A pointer to the endpoint descriptor table. This should be aligned at 512 byte address boundary. The size of the table is equal to DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed in the application.
uint32_t rootHubAvailableCurrent;	Root hub available current in milliamperes. This specifies the amount of current that root hub can provide to the attached device. This should be specified in mA. This is required when the driver is required to operate in host mode.
DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;	When operating in Host mode, the application can specify a Root Hub port enable function. This parameter should point to Root Hub port enable function. If this parameter is NULL, it implies that the Port is always enabled.
DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;	When operating in Host mode, the application can specify a Root Port Indication. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Root Port Indication is not supported.
DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;	When operating is Host mode, the application can specify a Root Port Overcurrent detection. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Overcurrent detection is not supported.

Description

USB Device Driver Initialization Data.

This structure contains all the data necessary to initialize the USB Driver. A pointer to a structure of this type, containing the desired initialization data, must be passed into the DRV_USBFS_Initialize function.

Remarks

None.

DRV_USBFS_OPMODES Enumeration

Identifies the operating modes supported by the USB Driver.

File

drv_usbfs.h

C

```
typedef enum {
  DRV_USBFS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE,
  DRV_USBFS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE,
  DRV_USBFS_OPMODE_HOST = DRV_USB_OPMODE_HOST,
  DRV_USBFS_OPMODE_OTG = DRV_USB_OPMODE_OTG
} DRV_USBFS_OPMODES;
```

Members

Members	Description
DRV_USBFS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE	The driver should be able to switch between host and device mode
DRV_USBFS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE	The driver should support device mode operation only
DRV_USBFS_OPMODE_HOST = DRV_USB_OPMODE_HOST	The driver should support host mode operation only
DRV_USBFS_OPMODE_OTG = DRV_USB_OPMODE_OTG	The driver should support the OTG protocol

Description

USB Operating Modes Enumeration.

This enumeration identifies the operating modes supported by the USB Driver.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_INDICATION Type

USB Root hub Application Hooks (Port Indication).

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_ROOT_HUB_PORT_INDICATION)(uint8_t port, USB_HUB_PORT_INDICATOR_COLOR color,
USB_HUB_PORT_INDICATOR_STATE state);
```

Description

USB Root hub Application Hooks (Port Indication).

A function of the type defined here should be provided to the driver root to implement Port Indication. The root hub driver calls this function when it needs to update the state of the port indication LEDs. The application can choose to implement the Amber and Green colors as one LED or two different LEDs. The root hub driver specifies the color and the indicator attribute (on, off or blinking) when it calls this function.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT Type

USB Root hub Application Hooks (Port Overcurrent detection).

File

drv_usbfs.h

C

```
typedef bool (* DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT)(uint8_t port);
```

Description

USB Root hub Application Hooks (Port Overcurrent detection).

A function of the type defined here should be provided to the driver root hub to check for port over current condition. This function will be called periodically by the root hub driver to check the Overcurrent status of the port. It should continue to return true while the Overcurrent condition exists on the port. It should return false when the Overcurrent condition does not exist.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE Type

USB Root hub Application Hooks (Port Power Enable/ Disable).

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE)(uint8_t port, bool control);
```

Description

USB Root hub Application Hooks (Port Power Enable/ Disable).

A function of the type defined here should be provided to the driver root to control port power. The root hub driver will call this function when it needs to enable port power. If the application circuit contains a VBUS switch, the switch should be accessed and controlled by this function. If the enable parameter is true, the switch should be enabled and VBUS should be available on the port. If the enable parameter is false, the switch should be disabled and VBUS should not be available on the port.

Remarks

None.

DRV_USBFS_DEVICE_INTERFACE Macro

USB Driver Device Mode Interface Functions.

File

drv_usbfs.h

C

```
#define DRV_USBFS_DEVICE_INTERFACE
```

Description

USB Driver Device Mode Interface Functions.

The Device Driver interface in the Device Layer Initialization data structure should be set to this value so that Device Layer can access the USB Driver Device Mode functions.

Remarks

None.

DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE Macro

USB Driver Endpoint Table Entry Size in bytes.

File

drv_usbfs.h

C

```
#define DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE 32
```

Description

USB Driver Endpoint Table Entry Size in bytes.

This constant defines the size (in bytes) of an entry in the endpoint table.

Remarks

None.

DRV_USBFS_HOST_INTERFACE Macro

USB Driver Host Mode Interface Functions.

File

drv_usbfs.h

C

```
#define DRV_USBFS_HOST_INTERFACE
```

Description

USB Driver Host Mode Interface Functions.

The Host Controller Driver interface in the Host Layer Initialization data structure should be set to this value so that Host Layer can access the USB Driver Host Mode functions.

Remarks

None.

DRV_USBFS_HOST_PIPE_HANDLE_INVALID Macro

Value of an Invalid Host Pipe Handle.

File

drv_usbfs.h

C

```
#define DRV_USBFS_HOST_PIPE_HANDLE_INVALID ((DRV_USBFS_HOST_PIPE_HANDLE)(-1))
```

Description

USB Driver Invalid Host Pipe Handle.

This constant defines the value of an Invalid Host Pipe Handle.

Remarks

None.

DRV_USBFS_INDEX_0 Macro

USB Driver Module Index 0 Definition.

File

drv_usbfs.h

C

```
#define DRV_USBFS_INDEX_0 0
```

Description

USB Driver Module Index 0 Definition.

This constant defines the value of USB Driver Index 0. The SYS_MODULE_INDEX parameter of the DRV_USBFS_Initialize and

DRV_USBFS_Open functions should be set to this value to identify instance 0 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBFS_Initialize and DRV_USBFS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

DRV_USBFS_INDEX_1 Macro

USB Driver Module Index 1 Definition.

File

drv_usbfs.h

C

```
#define DRV_USBFS_INDEX_1 1
```

Description

USB Driver Module Index 1 Definition.

This constant defines the value of USB Driver Index 1. The SYS_MODULE_INDEX parameter of the DRV_USBFS_Initialize and DRV_USBFS_Open functions should be set to this value to identify instance 1 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBFS_Initialize and DRV_USBFS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

Files

Files

Name	Description
drv_usbfs.h	PIC32MX USB Module Driver Interface File.
drv_usbfs_config_template.h	USB Full Speed (USBFS) Driver Configuration Template.

Description

drv_usbfs.h

PIC32MX USB Module Driver Interface File.

Enumerations

	Name	Description
	DRV_USBFS_EVENT	Identifies the different events that the USB Driver provides.
	DRV_USBFS_OPMODES	Identifies the operating modes supported by the USB Driver.

Functions

	Name	Description
	DRV_USBFS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBFS_Close	Closes an opened-instance of the USB Driver.
	DRV_USBFS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBFS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBFS_DEVICE_CurrentSpeedGet	This function returns the USB speed at which the device is operating.
	DRV_USBFS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.

	DRV_USBFS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBFS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBFS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBFS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBFS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBFS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBFS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.
	DRV_USBFS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBFS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBFS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBFS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBFS_HOST_PipeClose	Closes an open pipe.
	DRV_USBFS_HOST_PipeSetup	Open a pipe with the specified attributes.
	DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBFS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBFS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBFS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBFS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBFS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.
	DRV_USBFS_Initialize	Initializes the USB Driver.
	DRV_USBFS_Open	Opens the specified USB Driver instance and returns a handle to it.
	DRV_USBFS_Status	Provides the current status of the USB Driver module.
	DRV_USBFS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBFS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.

Macros

	Name	Description
	DRV_USBFS_DEVICE_INTERFACE	USB Driver Device Mode Interface Functions.
	DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE	USB Driver Endpoint Table Entry Size in bytes.
	DRV_USBFS_HOST_INTERFACE	USB Driver Host Mode Interface Functions.
	DRV_USBFS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBFS_INDEX_0	USB Driver Module Index 0 Definition.
	DRV_USBFS_INDEX_1	USB Driver Module Index 1 Definition.

Structures

	Name	Description
	DRV_USBFS_INIT	This type definition defines the Driver Initialization Data Structure.

Types

	Name	Description
	DRV_USBFS_EVENT_CALLBACK	Type of the USB Driver event callback function.
	DRV_USBFS_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe Handle type.
	DRV_USBFS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).

Description

PIC32MX USB Module Driver Interface Header File.

The PIC32MX Full speed USB Module driver provides a simple interface to manage the "USB" peripheral on PIC32MX microcontrollers. This file defines the interface definitions and prototypes for the USB driver. The driver interface meets the requirements of the MPLAB Harmony USB Host and Device Layer.

File Name

drv_usbfs.h

Company

Microchip Technology Inc.

drv_usbfs_config_template.h

USB Full Speed (USBFS) Driver Configuration Template.

Macros

	Name	Description
	DRV_USBFS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBFS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.
	DRV_USBFS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBFS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBFS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBFS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBFS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBFS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

USB Full Speed Driver Configuration Template.

This file lists all the configurations constants that affect the operation of the USBFS Driver.

File Name

drv_usbfs_config_template.h

Company

Microchip Technology Inc.

PIC32MZ USB Driver

Provides information on the USB Driver specific to PIC32MZ devices.

Description

The PIC32MZ USB Driver in MPLAB Harmony provides API functions that allow the MPLAB Harmony USB Host and Device Stack to access the

USB while operating on a PIC32MZ microcontroller. The driver implements the USB Driver Common Interface required by the USB Host and Device Stack. It abstracts the USB module operational details from the Host and Device Stack and provides the stacks with a modular access mechanism to the USB. The PIC32MZ USB Driver features the following:

- USB 2.0 High Speed and Full Speed operation in Peripheral mode
- USB 2.0 High Speed, Full Speed and Low Speed USB Peripheral Support in Host mode
- Designed for Dual Role Operation
- Capable of operating multiple USB modules
- Features non-blocking function and is interoperable with other MPLAB Harmony modules
- Features thread safe functions when operating within an RTOS
- Capable of operating in Polled and Interrupt modes
- Implements the USB Driver Common Interface required by the MPLAB Harmony USB Host and Device Stack
- Completely configurable through the MPLAB Harmony Configurator (MHC)
- Implements feature separation (Host and Device mode functions are implemented across different files)
- Designed to use the module's built-in DMA controller and transfer scheduler

Note: This help section only discusses features that are unique to the PIC32MZ USB Driver and are not a part of the USB Driver Common Interface. The driver functions that implement the USB Driver Common Interface are described in the Common Interface Help section.

While the PIC32MZ USB module supports USB "On-The-Go" (OTG), the PIC32MZ Driver does not currently implement USB OTG protocol support.

This help section only provides relevant information about the operation of the USB. The reader is encouraged to refer to the USB 2.0 Specification available at www.usb.org for a detailed explanation of USB protocol.

Using the Library

This topic describes the basic architecture of the USB PIC32MZ Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usbhs.h`

The interface to the PIC32MZ USB Driver library is defined in the `drv_usbhs.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Provides an overview of the library.

Description

The PIC32MZ USB Driver will typically be used by a USB Host and/or Device Stack. The USB Host and Device Stack operate as driver client applications. The driver is initialized as part of the MPLAB Harmony System Initialization. The driver initialization data structure specifies the operation mode (Host, Device, or Dual Role) of the driver. The driver features task routines to be called in the MPLAB Harmony application tasks function (SYS_Tasks function) and the USB Module Interrupt Service Routine (ISR).

The Host and the Device Stack can open the driver only when initialization has completed. It will continue to return an invalid driver handle while the initialization is in progress. Once opened, the Device Mode function can be called if the driver is operating in Device mode. The Host Mode function can be called if the driver is operating in Host mode. In Dual Role operation mode, the driver supports Host and Device operation in the same application. Even then, the driver will either operate as a USB Host or Device. OTG operation is not supported.

The PIC32MZ USB Driver features RTOS thread-safe functions. This allows the driver client application to safely call driver functions across different RTOS threads. Not all of the driver functions are interrupt-safe.

In addition to the USB Driver, which implements the USB Driver Common Interface, the PIC32MZ USB Driver implements functions which are required for its operation in the MPLAB Harmony framework. The following table lists the different categories of functions in the PIC32MZ USB Driver.

Library Interface Section	Description
System Function	These functions are accessed by the MPLAB Harmony System module. They allow the driver to be initialized, deinitialized and maintained. These functions are implemented in the `drv_usbhs.c` source file.
Client Core Functions	These functions allow the USB Host and Device Stack to open, close and perform other general driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs.c` source file.
Device Mode Operation Functions	These functions allow the USB Device Stack to perform USB Device mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_device.c` source file

Host Mode Operation Functions	These functions allow the USB Host Stack to perform USB Host mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_host.c` source file.
Root Hub Functions	These functions allow the USB Host Stack to access the driver Root hub operation. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_host.c` source file.

Abstraction Model

Provides information on the abstraction model for the library.

Description

The PIC32MZ USB Driver implements the abstraction model defined by the USB Driver Common interface. This interface abstracts USB module specific details and provides a module independent interface to the driver client applications.

While operating in Device mode, the driver expects the client application (the USB Device Stack) to enable endpoints and then submit I/O request packet (IRP) requests to the enabled endpoints. Multiple IRPs can be queued on an endpoint. The driver calls the IRP callback function when the IRP is processed. The driver allows the client application to also attach and detach the device on the bus. It generates events which indicate USB states.

While operating in Host mode, the driver expects the client application (the USB Host Stack) to open pipes to endpoints on the connected device. The client application can then submit IRPs to the pipes. Multiple IRPs can be queued on a pipe. The driver calls the IRP callback function when the IRP is processed. The driver will call application defined functions to enumerate and denumerate a device. These functions are called when the driver detect device attach and detach respectively. The driver also exports root hub functions to the client application. This allows the client application to treat the driver as a single port hub

Please refer to the PIC32 USB Driver Common Interface help section for more details on the driver abstraction model.

How the Library Works

Provides information on how the library works.

Description

This section only explains aspects of driver operation which are unique to the PIC32MZ USB Driver. Major driver operations are described in the PIC32 USB Driver Common Interface help section.

Driver Initialization

Note: While generating a MPLAB Harmony USB project with MHC, the initialization code for the driver is generated automatically based on selections made in the USB Host stack or Device Stack Configuration trees.

The PIC32MZ USB Driver must be initialized so that a client application can open. The client application will not be able to open the driver if the initialization is in progress or has failed. The driver is initialized by calling the DRV_USBHS_Initialize function. This function is called from the SYS_Initialize function in the MPLAB Harmony application project and accepts two input parameters. The `index` parameter defines the instance of the USB Driver to be initialized. This becomes significant when the PIC32MZ microcontroller has more than one USB module. The `init` parameter is a driver-specific data structure of the type DRV_USBHS_INIT. This structure is shown in the following code example.

```
/* This code show the PIC32MZ USB Driver Initialization data structure.
 * A structure of this type must be provided to the DRV_USBHS_Initialize
 * function. */

typedef struct
{
    /* System Module Initialization */
    SYS_MODULE_INIT moduleInit;

    /* Identifies the USB peripheral to be used. This should be the USB PLIB
       module instance identifier. */
    uint8_t usbID;

    /* This should be set to true if the USB module must stop operation in Idle
       mode */
    bool stopInIdle;

    /* This should be set to true if the USB module must suspend when the CPU
       enters Sleep mode. */
    bool suspendInSleep;

    /* Specify the interrupt source for the USB module. This should be Interrupt
```

```
    PLIB Interrupt source identifier for the USB module instance specified in
    usbID. */
INT_SOURCE interruptSource;

/* Specify the interrupt source for the USB module specific DMA controller.
 * This should be the PLIB Interrupt source identified for the USB
 * module instance specified in usbID. */
INT_SOURCE interruptSourceUSBDma;

/* Specify the operational speed of the USB module. This should always be
   set to USB_SPEED_FULL. */
USB_SPEED operationSpeed;

/* Specify the operation mode of the USB module. This defines if the USB
 * module will support Device, Host or Dual Role operation */
DRV_USBHS_OPMODES operationMode;

/* A pointer to the endpoint descriptor table. This should be aligned at 512
   byte address boundary. The size of the table is equal to the
   DRV_USBHS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed
   in the application. */
void * endpointTable;

/* Root hub available current in mA. This specifies the amount of current
   that root hub can provide to the attached device. This should be
   specified in mA. This is required when the driver is required to operate
   in host mode. */
uint32_t rootHubAvailableCurrent;

/* When operating in Host mode, the application can specify a Root Hub port
   enable function. This parameter should point to Root Hub port enable
   function. If this parameter is NULL, it implies that the Port is always
   enabled. */
DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;

/* When operating in Host mode, the application can specify a Root Port
   Indication. This parameter should point to the Root Port Indication
   function. If this parameter is NULL, it implies that Root Port Indication
   is not supported. */
DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;

/* When operating is Host mode, the application can specify a Root Port
   Overcurrent detection. This parameter should point to the Root Port
   Indication function. If this parameter is NULL, it implies that
   Overcurrent detection is not supported. */
DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;

} DRV_USBHS_INIT;
```

The operationMode parameter defines the driver operation mode. This can be set to DRV_USBFS_OPMODE_DEVICE, DRV_USBFS_OPMODE_HOST, or DRV_USBFS_OPMODE_DUAL_ROLE for Device, Host and Dual Role operation, respectively.

The `rootHubAvailableCurrent` parameter should be set to the maximum current that the VBUS power supply can provide on the bus. The driver does not use this information directly. It provides this data to the client application while operating in Host mode.

The `portPowerEnable` parameter must point to a Port Power Enable function. The driver, while operating in Host mode, will call this function to enable the VBUS switch. This function should activate the VBUS switch if the driver calls this function with the enable parameter set to true. It should deactivate the switch if the driver calls this function with the `enable` parameter set to false. This parameter should be set to NULL if such a switch (of the switch control) is not available in the application.

The `portIndication` parameter must point to a Port Indication function. The driver, while operating in Host mode, will call this function to indicate the current state of the port. The driver will call this function with LED color status as defined in Chapter 11 of the USB 2.0 Specification. This parameter should be set to NULL if such a LED indication is not available in the application.

The `portOverCurrentDetect` parameter must point to a Port Overcurrent Detect function. The driver, while operating in Host mode, will call this function periodically to check if the attached device is overdrawing current. If the function should return true if such a condition exists. This parameter should be set to NULL if such detection is not available in the application.

The following code example shows initialization of the driver for Device mode operation.

```
/* This code shows an example of DRV_USBHS_INIT data structure for
 * Device mode operation. Here the driver is initialized to work with USB0 USB
 * module. */
```

```
DRV_USBHS_INIT init;
SYS_MODULE_OBJ usbDriverObj;

const DRV_USBHS_INIT drvUSBInit =
{
    /* Interrupt Source for USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* DMA Interrupt Source for USB module */
    .interruptSourceUSBDma = INT_SOURCE_USB_1_DMA,

    /* System module initialization */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Module operate in device mode */
    .operationMode = DRV_USBHS_OPMODE_DEVICE,

    /* Module operated at high speed */
    .operationSpeed = USB_SPEED_HIGH,

    /* Stop in idle */
    .stopInIdle = false,

    /* Suspend in sleep */
    .suspendInSleep = false,

    /* Identifies peripheral (PLIB-level) ID */
    .usbID = USBHS_ID_0
};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecast to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBHS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBHS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBHS_Tasks() takes the driver module object returned by the
     * DRV_USBHS_Initialize funciton as a parameter. */

    DRV_USBHS_Tasks(usbDriverObj);
}

void __ISR(_USB_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBHS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for Polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBHS_Tasks_ISR(usbDriverObj);
}

void __ISR ( _USB_DMA_VECTOR,ipl4AUTO) _IntHandlerUSBInstance0_USBDMA ( void )
{
    DRV_USBHS_Tasks_ISR_USBDMA(usbDriverObj);
}
```

The following code example shows initialization of the driver for Host mode operation.

```
/* This code shows an example of how the Hi-Speed USB (USBHS) driver can be configured
 * for Host mode operation. In this example, the
 * BSP_USBVBUSSwitchOverCurrentDetect function checks for over current condition
 * and the BSP_USBVBUSPowerEnable function enables the VBUS power. The port
```

```
    * indication function is not implemented and hence the portIndication member of
    * the initialization data structure is set to NULL. */

   /* The implementation of the port over current detect, indication and the VBUS
    * power supply functions is discussed later in this help section. */

   DRV_USBHS_INIT drvUSBHSInit =
   {
       /* This should always be set to SYS_MODULE_POWER_RUN_FULL. */
       .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

       /* Interrupt Source for the USB module */
       .interruptSource = INT_SOURCE_USB_1,

       /* Interrupt Source for the USB DMA module */
       .interruptSourceUSBDma = INT_SOURCE_USB_1_DMA,

       /* Configure for host mode operation. */
       .operationMode = DRV_USBHS_OPMODE_HOST,

       /* The driver should run at high speed. */
       .operationSpeed = USB_SPEED_HIGH,

       /* Port indication function is not implemented and is not available */
       .portIndication = NULL,

       /* This is the VBUS Power enable function */
       .portPowerEnable = BSP_USBVBUSPowerEnable,

       /* This is the over current detect function. */
       .portOverCurrentDetect = BSP_USBVBUSSwitchOverCurrentDetect,

       /* Here we state that the VBUS power supply can provide at most 500 mA of
        * current */
       .rootHubAvailableCurrent = 500,

       /* Moudule will operate in IDLE. */
       .stopInIdle = false,

       /* Module will not suspend automatically in sleep */
       .suspendInSleep = false,

       /* USB Module ID is 1 */
       .usbID = USBHS_ID_0

   };

   void SYS_Initialize(void)
   {
       /* Initialize the USB Driver. Note how the init parameter is typecast to
        * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
        * is passed to the driver tasks routine. DRV_USBHS_INDEX_0 is helper
        * constant defined in drv_usbfs.h */

       usbDriverObj = DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
   }

   void SYS_Tasks(void)
   {
       /* The polled state of the USB driver is updated by calling the
        * DRV_USBHS_Tasks function in the SYS_Tasks() function. The
        * DRV_USBHS_Tasks takes the driver module object returned by the
        * DRV_USBHS_Initialize funciton as a parameter. */

       DRV_USBHS_Tasks(usbDriverObj);
   }

   void __ISR( _USB_VECTOR , IPL4AUTO)_IntHandler_USB_stub ( void )
   {
```

```
    /* The DRV_USBHS_Tasks_ISR function updates the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBHS_Tasks_ISR(usbDriverObj);
}

void __ISR ( _USB_DMA_VECTOR, IPL4AUTO)  _IntHandlerUSBInstance0_USBDMA ( void )
{
    /* The DRV_USBHS_Tasks_ISR_USBDMA function update the DMA transfer state of
     * the USB Driver. */

     DRV_USBHS_Tasks_ISR_USBDMA(usbDriverObj);
}
```

The PIC32MX USB Driver requires definition of configuration constants to be available in the `system_config.h` file of the MPLAB Harmony Application Project Configuration. Refer to the Configuring the Library section for details.

Multi-client Operation

The PIC32MZ USB Driver supports multi-client operation. In that, it can be opened by two application clients. This is required where Dual Operation is desired. The following should be noted when using multi-client operation:

- The driver should be initialized for Dual Role Operation mode.
- The DRV_USBHS_Open function can be called at the most twice in the application. The driver supports a maximum of two clients.
- A client can access either the host or device functionality of the driver. It cannot do both.
- It is possible for the two clients to operate in two different threads while operating with an RTOS.

Note: The typical the application clients for PIC32MZ USB Driver would be the MPLAB Harmony USB Host and Device Stack. The complexity of operating the driver in Dual Role mode is handled by the stack operation. The MHC will configure the driver for Dual Role operation when such operation is selected in USB Stack configuration tree.

USB Driver Common Interface

The PIC32MZ USB Driver exports its implementation of the USB Driver Common Interface to the Host and Device Layer via the DRV_USBHS_HOST_INTERFACE and DRV_USBHS_DEVICE_INTERFACE structures. The DRV_USBHS_HOST_INTERFACE structure is defined in the `drv_usbhs_host.c` file. The following code example shows this structure.

```
/*********************************************************
 * This structure is a set of pointer to the USBHS driver
 * functions. It is provided to the host and device layer
 * as the interface to the driver.
 * *******************************************************/

DRV_USB_HOST_INTERFACE gDrvUSBHSHostInterface =
{
    .open = DRV_USBHS_Open,
    .close = DRV_USBHS_Close,
    .eventHandlerSet = DRV_USBHS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBHS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBHS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBHS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBHS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBHS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBHS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBHS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet = DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBHS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBHS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBHS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBHS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

The DRV_USBFS_DEVICE_INTERFACE structure is defined in the `drv_usbhs_device.c` file. The following code example shows this structure.

The MPLAB Harmony USB Host and Device stack perform driver independent access through the function pointers contained in these structures.

```
/*************************************************
```

```
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * device layer when the device layer must use the
 * PIC32MZ USB Controller.
 ************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBHSDeviceInterface =
{
    .open = DRV_USBHS_Open,
    .close = DRV_USBHS_Close,
    .eventHandlerSet = DRV_USBHS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBHS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBHS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBHS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBHS_DEVICE_Attach,
    .deviceDetach = DRV_USBHS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBHS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBHS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBHS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBHS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBHS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBHS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBHS_DEVICE_IRPSubmit,
    .deviceIRPCancelAll = DRV_USBHS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBHS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBHS_DEVICE_RemoteWakeupStart,
    .deviceTestModeEnter = DRV_USBHS_DEVICE_TestModeEnter


};
```

Operation with RTOS

The PIC32MZ USB Driver is designed to operate with a RTOS. The driver implementation uses the MPLAB Harmony Operating System Abstraction Layer (OSAL). This allows the driver to function with entire range of RTOSes supported in MPLAB Harmony. The following points must be considered while using the driver with an RTOS.

- The driver can be opened from different threads
- In Device mode, an enabled endpoint should only be accessed from one thread. For example, if an application requires two endpoints, Endpoint 2 and Endpoint 3, the application could contain two threads, one accessing Endpoint 2 and another accessing Endpoint 3. The thread accessing Endpoint 2 cannot access Endpoint 3.
- While operating in Host mode, endpoint pipes can be opened from different threads. A pipe handle to an open pipe cannot be shared across threads.

USB DMA Operation

The PIC32MZ USB module features a built-in DMA controller. This controller works independently of the PIC32MZ DMA controller. The PIC32MZ USB Driver uses USB DMA controller to expedite transfer of memory from the USB module FIFO to user application memory. The following should be noted for the USB DMA controller:

- If the PIC32MZ USB Driver could not allocate a DMA channel (all channels are busy), it will use the CPU instructions to unload the endpoint FIFOs
- The USB module and the USB DMA controller interrupt priorities should be the same
- The application buffer start address should always be aligned on a 16-byte boundary and should be placed in coherent memory. Refer to the description of the DRV_USBHS_HOST_IRPSubmit and DRV_USBHS_DEVICE_IRPSubmit functions for details on how the user application buffer should be allocated.

Root Hub Operation

The PIC32MZ USB Driver implements a Root Hub Driver Interface. This allows the driver to emulate a hub. The USB Host Stack enumerates the Root Hub as a device. The Host Stack then does not differentiate between an external hub and the root hub. While emulating a hub, the PIC32MZ USB Driver Root Hub appears as a single port hub.

As a part of the Root Hub interface, the PIC32MZ USB Driver requires the application to supply functions for hub features that it does not implement. These features are:

- Port Overcurrent Detect
- VBUS Switch Control
- Port Indication

A pointer to these functions (if implemented) must be supplied through the driver initialization data (of the type DRV_USBHS_INIT) structure at the time of driver initialization. The application has the option of not implementing these functions. In such a case, the function pointers for the unimplemented function, in the initialization data structure should be set to NULL.

The root hub driver must also be able to communicate the maximum current capability of its port to the USB Host Layer. The PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for controlling the root hub port current. To facilitate this request, the driver will report the current capability that was specified in the `rootHubAvailableCurrent` parameter of the driver initialization data structure. The application must set this parameter to report the current supply capability of the VBUS power supply. The USB Host Layer uses this value to manage the bus current budget. If a connected device reports a configuration that requires more current than what the VBUS power supply can provide, the host will not set the configuration.

Port Overcurrent Detect

The Root Hub operation in PIC32MZ USB Driver will periodically call a Port Overcurrent Detect function to detect if an overcurrent condition is active on the port. The application must supply this function if port overcurrent detection is needed. The PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for checking overcurrent condition. The overcurrent condition on the port can occur in a case where the attached device has malfunctioned or when the USB VBUS line has short circuited to ground.

The signature of the function and an example implementation is shown in the following code example. The function must return (and must continue to return) true if an overcurrent condition exists on the port.

```
/* This code shows an example implementation of the
 * portOverCurrentDetect function. The PIC32MZ USB Driver will call this
 * function periodically to check if an over current condition exists on the
 * port. In this example, we assume that the over current detect pin from an
 * external circuit in the system, is connected to  port RD0 and the pin logic
 * is active high. The function must return true if an over current condition is
 * present on this pin */


bool BSP_USBVBUSSwitchOverCurrentDetect(uint8_t port)
{
    if(PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_D, 0) == 1)
    {
        return(true);
    }
    else
    {
        return(false);
    }
}
```

VBUS Switch Control

The PIC32MZ USB Driver Root Hub operation will attempt to control the VBUS power supply to the port. Because the PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for checking controlling VBUS, such a control function must be supplied by the application. The root hub operation will access this function when the PIC32MX USB Driver will call the portPowerEnable function as a part of the Bus Enable sequence.

The following code shows an example of how this function can be implemented.

```
/* This code shows an example implementation of the VBUS Power Enable
 * function. The PIC32MZ USB Driver will call this function as a part of bus
 * enable function. In this example, it is assumed that system contains an
 * external VBUS power switch and this is control by port RB5.
 */

void BSP_USBVBUSPowerEnable(uint8_t port, bool enable)
{
    if(enable)
    {
        PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
    else
    {
        PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
}
```

Port Indication Function

The Root Hub Operation in the PIC32MZ USB Driver allows display of Port LED status. If the application requires this indication, it must implement a function which the Root Hub operation would call when a change in the Root Hub port has occurred. The port indication operation is specified in Section 11.5.3 of the USB 2.0 Specification.

```
/* This code shows an example implementation of the port indication
 * function. The PIC32MZ USB Driver call this function when it wants to indicate
 * port status. It is assumed that three function to switch off, blink and
 * switch on an LED are available. It is further assumed that these function
```

```
 * accept the color of the LED to operated on. */

void BSP_RootHubPortIndication
(
    uint8_t port,
    USB_HUB_PORT_INDICATOR_COLOR color,
    USB_HUB_PORT_INDICATOR_STATE state
)
{
    /* The color parameter indicates the color of the LED to be affected. The
     * color will be either USB_HUB_PORT_INDICATOR_COLOR_GREEN or
     * USB_HUB_PORT_INDICATOR_COLOR_AMBER. */

    switch (state)
    {
        case USB_HUB_PORT_INDICATOR_STATE_OFF:
            BSP_SwitchLEDOff(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_BLINKING:
            BSP_LEDBlink(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_ON:
            BSP_SwitchLEDOn(color);
            break;
        default:
            break;
    }
}
```

Configuring the Library

Provides information on the configuring the library.

Macros

	Name	Description
	DRV_USBHS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBHS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.
	DRV_USBHS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBHS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBHS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBHS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBHS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBHS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

The PIC32MZ USB Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_USBHS_DEVICE_SUPPORT Macro

Determines if the USB Device Functionality should be enabled.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_DEVICE_SUPPORT true
```

Description

Hi-Speed USB Driver Device Mode Support.

This constant should be set to true if USB device support is required in the application. It should be set to false if device support is not required.

Remarks

This constant should always be defined.

DRV_USBHS_ENDPOINTS_NUMBER Macro

Configures the number of endpoints to be provisioned in the driver.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_ENDPOINTS_NUMBER 3
```

Description

Hi-Speed USB Driver Endpoint Numbers.

This constant configures the number of endpoints that the driver needs to manage. When DRV_USBHS_DEVICE_SUPPORT is enabled, this constant should be set to the total number of endpoints to be enabled in the device. When enabled, an endpoint can be used for communication. Using any direction of an endpoint will require that the entire endpoint to be enabled.

Consider the case of a composite USB Device that contains a CDC and MSD function. The CDC function will require one Bulk endpoint (OUT and IN directions) and one Interrupt endpoint (IN direction). The MSD function will require one Bulk endpoint (IN and OUT directions). This design can be implemented by using four endpoints. Endpoint 0 is used for the mandatory control interface. Endpoint 1 is used for CDC Bulk interface. Endpoint 2 is used for CDC Interrupt interface and Endpoint 3 is used for MSD Bulk Interface. The constant should then be set to 4.

For Host mode operation, this constant should be set to 1. Setting this value to greater than 1 will result in unused data memory allocation.

Remarks

This constant should always be defined.

DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION Macro

Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION 500
```

Description

Hi-Speed USB Driver Host Mode Attach Debounce Duration.

This constant configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach. When the driver first detects a device attach, it will start a timer for the duration specified by the constant. When the timer expires, the driver will check if the device is still attached. If so, the driver will then signal an attach event to the host stack. The duration allows for the device attach to become electro-mechanically stable.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_NAK_LIMIT Macro

Configures the NAK Limit for Host Mode Control Transfers.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_NAK_LIMIT 2000
```

Description

Hi-Speed USB Driver Host Mode Control Transfers NAK Limit.

This constant configures the number of NAKs that the driver can accept from the device in the data stage of a control transfer before aborting the control transfer with a USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT. Setting this constant to 0 will disable NAK limit checking. This constant should be adjusted to enable USB host compatibility with USB Devices that require more time to process control transfers.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_PIPES_NUMBER Macro

Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_PIPES_NUMBER 10
```

Description

Hi-Speed USB Driver Host Mode Pipes Number.

This constant configures the maximum number of pipes that can be opened when the driver is operating in Host mode. Calling the DRV_USBHS_HOST_PipeSetup function will cause a pipe to be opened. Calling this function when DRV_USBHS_HOST_PIPES_NUMBER number of pipes have already been opened will cause the function to return an Invalid Pipe Handle. This constant should be configured to account for the maximum number of devices and the device types to be supported by the host application.

For example, if the USB Host application must support two USB Mass Storage devices and one CDC device. A CDC device requires four pipes and a Mass Storage Device requires three pipes. This constant should therefore be set to a value of 9 (four bulk pipes for two Mass Storage devices + two bulk pipes and one Interrupt pipe for one CDC device and two control pipes for two devices). Allocating pipes consumes data memory.

While enabling support for multiple devices, through a Hub, the application should consider the worst case requirement while configuring this constant. For example, a case where devices with the most number of pipe requirements are connected to the hub. At the same time, setting this constant to more than what is required will consume data memory.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_RESET_DURATION Macro

Configures the time duration (in milliseconds) of the Reset Signal.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_RESET_DURATION 100
```

Description

Hi-Speed USB Driver Host Mode Reset Duration.

This constant configures the duration of the reset signal. The driver generates a reset signal when the USB Host stack requests for a root hub port reset. The driver will generate the reset signal for the duration specified by this constant and will then stop generating the reset signal.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_SUPPORT Macro

Determines if the USB Host Functionality should be enabled.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_SUPPORT false
```

Description

Hi-Speed USB Driver Host Mode Support.

This constant should be set to true if USB Host mode support is required in the application. It should be set to false if host support is not required.

Remarks

This constant should always be defined.

DRV_USBHS_INSTANCES_NUMBER Macro

Specifies the number of driver instances to be enabled in the application.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_INSTANCES_NUMBER 1
```

Description

Hi-Speed USB Driver Instances Number.

This constant defines the number of driver instances to be enabled in the application. This will be typically be the number of USB controllers to be used in the application. On PIC32MZ microcontrollers that have one USB controller, this value will always be 1. On PIC32MZ microcontrollers that have two USB controllers, this value could be one or two, depending on whether one or two USB segments are required. To conserve data memory, this constant should be set to exactly the number of USB controllers that are required in the system.

Remarks

This constant should always be defined.

DRV_USBHS_INTERRUPT_MODE Macro

Configures the driver for interrupt or polling mode operation.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_INTERRUPT_MODE true
```

Description

Hi-Speed USB Driver Interrupt Mode.

This constant configures the driver for interrupt or polling operation. If this flag is set to true, the driver will operate in Interrupt mode. If the flag is set to false, the driver will operate in Polled mode. In Polled mode, the driver interrupt state machine gets updated in the SYS_Tasks function. If the driver is configured for Interrupt mode, the driver Interrupt state machine gets updated in the driver Interrupt Service Routine(ISR). It is always recommended for the driver to operate in Interrupt mode.

Remarks

This constant should always be defined.

Building the Library

This section lists the files that are available in the PIC32MZ USB Driver Library.

Description

This section list the files that are available in the `\src` folder of the PIC32MZ USB Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usb/usbhs`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_usbhs.h	This file should be included by any .c file which accesses the PIC32MZ USB Driver API. This one file contains the prototypes for all driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_usbhs.c	This file should always be included in the project when using the PIC3MZ USB Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/dynamic/drv_usbhs_device.c	This file should be included in the project if Device mode operation is required.
/src/dynamic/drv_usbhs_host.c	This file should be included in the project if Host mode operation is required.

Module Dependencies

The PIC32MZ USB Driver Library depends on the following modules:

- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_USBHS_Initialize	Initializes the Hi-Speed USB Driver.
	DRV_USBHS_Status	Provides the current status of the Hi-Speed USB Driver module.
	DRV_USBHS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBHS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.
	DRV_USBHS_Tasks_ISR_USBDMA	Maintains the driver's DMA Transfer state machine and implements its ISR.

b) Client Core Functions

	Name	Description
	DRV_USBHS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBHS_Close	Closes an opened-instance of the Hi-Speed USB Driver.
	DRV_USBHS_Open	Opens the specified Hi-Speed USB Driver instance and returns a handle to it.

c) Device Mode Operation Functions

	Name	Description
	DRV_USBHS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBHS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBHS_DEVICE_CurrentSpeedGet	This function will return the USB speed at which the device is operating.
	DRV_USBHS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBHS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBHS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBHS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBHS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.

	DRV_USBHS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBHS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.
	DRV_USBHS_DEVICE_TestModeEnter	This function enables the specified USB 2.0 Test Mode.
	DRV_USBHS_DEVICE_TestModeExit	This function disables the specified USB 2.0 Test Mode.

d) Host Mode Operation Functions

	Name	Description
	DRV_USBHS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBHS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBHS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBHS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBHS_HOST_PipeClose	Closes an open pipe.
	DRV_USBHS_HOST_PipeSetup	Open a pipe with the specified attributes.

e) Root Hub Functions

	Name	Description
	DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBHS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBHS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBHS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBHS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBHS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.

f) Data Types and Constants

	Name	Description
	DRV_USBHS_EVENT	Identifies the different events that the Hi-Speed USB Driver provides.
	DRV_USBHS_EVENT_CALLBACK	Type of the Hi-Speed USB Driver event callback function.
	DRV_USBHS_HOST_PIPE_HANDLE	Defines the Hi-Speed USB Driver Host Pipe Handle type.
	DRV_USBHS_INIT	This type definition defines the Driver Initialization Data Structure.
	DRV_USBHS_OPMODES	Identifies the operating modes supported by the Hi-Speed USB Driver.
	DRV_USBHS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).
	DRV_USBHS_DEVICE_INTERFACE	Hi-Speed USB Driver Device Mode Interface Functions.
	DRV_USBHS_HOST_INTERFACE	Hi-Speed USB Driver Host Mode Interface Functions.
	DRV_USBHS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBHS_INDEX_0	Hi-Speed USB Driver Module Index 0 Definition.

Description

This section describes the functions of the PIC32MZ USB Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_USBHS_Initialize Function

Initializes the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
SYS_MODULE_OBJ DRV_USBHS_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

- SYS_MODULE_OBJ_INVALID - The driver initialization failed.
- A valid System Module Object - The driver initialization was able to start. It may have not completed and requires the DRV_USBHS_Tasks function to be called periodically. This value will never be the same as SYS_MODULE_OBJ_INVALID.

Description

This function initializes the Hi-Speed USB Driver, making it ready for clients to open. The driver initialization does not complete when this function returns. The DRV_USBHS_Tasks function must called periodically to complete the driver initialization. The DRV_USBHS_Open function will fail if the driver was not initialized or if initialization has not completed.

Remarks

This function must be called before any other Hi-Speed USB Driver function is called. This function should only be called once during system initialization unless DRV_USBHS_Deinitialize is called to deinitialize the driver instance.

Preconditions

None.

Example

```
 // The following code shows an example initialization of the
 // driver. The USB module to be used is USB1.  The module should not
 // automatically suspend when the  microcontroller enters Sleep mode.  The
 // module should continue  operation when the module enters Idle mode.  The
 // power state is set to run at full clock speeds. Device Mode operation
 // should be at FULL speed. The size of the endpoint table is set for two
 // endpoints.

DRV_USBHS_INIT moduleInit;

usbInitData.usbID               = USBHS_ID_0;
usbInitData.opMode              = DRV_USBHS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

Parameters

Parameters	Description
drvIndex	Ordinal number of driver instance to be initialized. This should be set to DRV_USBHS_INDEX_0 if driver instance 0 needs to be initialized.
init	Pointer to a data structure containing data necessary to initialize the driver. This should be a DRV_USBHS_INIT structure reference typecast to SYS_MODULE_INIT reference.

Function

SYS_MODULE_OBJ DRV_USBHS_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT * const init
)

DRV_USBHS_Status Function

Provides the current status of the Hi-Speed USB Driver module.

File

drv_usbhs.h

C

```
SYS_STATUS DRV_USBHS_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is ready.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has never been initialized.

Description

This function provides the current status of the Hi-Speed USB Driver module.

Remarks

None.

Preconditions

The DRV_USBHS_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBHS_Initialize
SYS_STATUS          status;
DRV_USBHS_INIT moduleInit;

usbInitData.usbID                = USBHS_ID_0;
usbInitData.opMode               = DRV_USBHS_OPMODE_DEVICE;
usbInitData.stopInIdle           = false;
usbInitData.suspendInSleep       = false;
usbInitData.operationSpeed       = USB_SPEED_FULL;
usbInitData.interruptSource      = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);

// The status of the driver can be checked.
status = DRV_USBHS_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USBHS_Initialize function.

Function

SYS_STATUS DRV_USBHS_Status (SYS_MODULE_OBJ object)

DRV_USBHS_Tasks Function

Maintains the driver's state machine when the driver is configured for Polled mode.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Maintains the driver's Polled state machine. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks). This function will never block.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBHS_Initialize

while (true)
{
    DRV_USBHS_Tasks(object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize function).

Function

void DRV_USBHS_Tasks(SYS_MODULE_OBJ object)

DRV_USBHS_Tasks_ISR Function

Maintains the driver's Interrupt state machine and implements its ISR.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal Interrupt state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function should be called from the USB ISR. For multiple USB modules, it should be ensured that the correct Hi-Speed USB Driver system module object is passed to this function.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBHS_Initialize

while (true)
{
    DRV_USBHS_Tasks_ISR (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize).

Function

void DRV_USBHS_Tasks_ISR(SYS_MODULE_OBJ object)

DRV_USBHS_Tasks_ISR_USBDMA Function

Maintains the driver's DMA Transfer state machine and implements its ISR.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks_ISR_USBDMA(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal DMA Transfer state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function should be called from the USB DMA ISR. For multiple USB modules, it should be ensured that the correct Hi-Speed USB Driver system module object is passed to this function.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBHS_Initialize

while (true)
{
    DRV_USBHS_Tasks_ISR_USBDMA (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize).

Function

void DRV_USBHS_Tasks_ISR_USBDMA(SYS_MODULE_OBJ object)

b) Client Core Functions

DRV_USBHS_ClientEventCallBackSet Function

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.

File

drv_usbhs.h

C

```
void DRV_USBHS_ClientEventCallBackSet(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK
myEventCallBack);
```

Returns

None.

Description

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events. The callback is disabled by either not calling this function after the DRV_USBHS_Open function has been called or by setting the myEventCallBack argument as NULL. When the callback function is called, the hReferenceData argument is returned.

Remarks

Typical usage of the Hi-Speed USB Driver requires a client to register a callback.

Preconditions

None.

Example

```
// Set the client event callback for the Device Layer.  The
// USBDeviceLayerEventHandler function is the event handler. When this
// event handler is invoked by the driver, the driver returns back the
// second argument specified in the following function (which in this case
// is the Device Layer data structure). This allows the application
// firmware to identify, as an example, the Device Layer object associated
// with this callback.

DRV_USBHS_ClientEventCallBackSet(myUSBDevice.usbDriverHandle, (uintptr_t)&myUSBDevice,
USBDeviceLayerEventHandler);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
hReferenceData	Object (could be a pointer) that is returned with the callback.
myEventCallBack	Callback function for all USB events.

Function

void DRV_USBHS_ClientEventCallBackSet

(

DRV_HANDLE handle,

uintptr_t hReferenceData,

DRV_USBHS_EVENT_CALLBACK myEventCallBack

);

DRV_USBHS_Close Function

Closes an opened-instance of the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
void DRV_USBHS_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the Hi-Speed USB Driver, invalidating the handle.

Remarks

After calling this function, the handle passed in handle parameter must not be used with any of the other driver functions. A new handle must be obtained by calling DRV_USBHS_Open function before the caller may use the driver again.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance. DRV_USBHS_Open function must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USBHS_Open

DRV_USBHS_Close(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_Close(DRV_HANDLE handle)

DRV_USBHS_Open Function

Opens the specified Hi-Speed USB Driver instance and returns a handle to it.

File

drv_usbhs.h

C

```
DRV_HANDLE DRV_USBHS_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE_INVALID - The driver could not be opened successfully.This can happen if the driver initialization was not complete or if an internal error has occurred.
- A Valid Driver Handle - This is an arbitrary value and is returned if the function was successful. This value will never be the same as DRV_HANDLE_INVALID.

Description

This function opens the specified Hi-Speed USB Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The intent flag should always be DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NON_BLOCKING. Any other setting of the intent flag will return a invalid driver handle. A driver instance can only support one client. Trying to open a driver that has an existing client will result in an unsuccessful function call.

Remarks

The handle returned is valid until the DRV_USBHS_Close function is called. The function will typically return DRV_HANDLE_INVALID if the driver was not initialized. In such a case the client should try to open the driver again.

Preconditions

Function DRV_USBHS_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

// This code assumes that the driver has been initialized.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);
```

```
if(DRV_HANDLE_INVALID == handle)
{
    // The application should try opening the driver again.
}
```

Parameters

Parameters	Description
drvIndex	Identifies the driver instance to be opened. As an example, this value can be set to DRV_USBHS_INDEX_0 if instance 0 of the driver has to be opened.
intent	Should always be (DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE| DRV_IO_INTENT_NON_BLOCKING).

Function

DRV_HANDLE DRV_USBHS_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent
)

c) Device Mode Operation Functions

DRV_USBHS_DEVICE_AddressSet Function

This function will set the USB module address that is obtained from the Host.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);
```

Returns

None.

Description

This function will set the USB module address that is obtained from the Host in a setup transaction. The address is obtained from the SET_ADDRESS command issued by the Host. The primary (first) client of the driver uses this function to set the module's USB address after decoding the setup transaction from the Host.

Remarks

None.

Preconditions

None.

Example

```
// This function should be called by the first client of the driver,
// which is typically the Device Layer. The address to set is obtained
// from the Host during enumeration.

DRV_USBHS_DEVICE_AddressSet(deviceLayer, 4);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
address	The address of this module on the USB bus.

Function

void DRV_USBHS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);

DRV_USBHS_DEVICE_Attach Function

This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_Attach(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables the pull-up resistors on the D+ or D- lines thus letting the USB Host know that a device has been attached on the bus . This function should be called when the driver client is ready to receive communication from the Host (typically after all initialization is complete). The USB 2.0 specification requires VBUS to be detected before the data line pull-ups are enabled. The application must ensure the same.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBHS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be attached when VBUS Session Valid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBHS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBHS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBHS_DEVICE_Attach(handle);
            break;

        case DRV_USBHS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBHS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_Attach(DRV_HANDLE handle);

DRV_USBHS_DEVICE_CurrentSpeedGet Function

This function will return the USB speed at which the device is operating.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);
```

Returns

Returns a member of the USB_SPEED type.

Description

This function will return the USB speed at which the device is operating.

Remarks

None.

Preconditions

Only valid after the device is attached to the Host and Host has completed reset signaling.

Example

```
// Get the current speed.

USB_SPEED deviceSpeed;

deviceSpeed = DRV_USBHS_DEVICE_CurrentSpeedGet(deviceLayer);

if(deviceLayer == USB_SPEED_HIGH)
{
    // Possibly adjust buffers for higher throughput.
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

USB_SPEED DRV_USBHS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);

DRV_USBHS_DEVICE_Detach Function

This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_Detach(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables the pull-up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to Host mode operation). A self-powered device should be detached from the bus when the VBUS is not valid.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBHS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be detached when VBUS Session Invalid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBHS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBHS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBHS_DEVICE_Attach(handle);
            break;

        case DRV_USBHS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBHS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_Detach(DRV_HANDLE handle);

DRV_USBHS_DEVICE_EndpointDisable Function

This function disables an endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointDisable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The endpoint that is being accessed is not a valid endpoint (endpoint was not provisioned through the DRV_USBHS_ENDPOINTS_NUMBER configuration constant) defined for this driver instance.

Description

This function disables an endpoint. If the endpoint type is a control endpoint type, both directions are disabled. For non-control endpoints, the function disables the specified direction only. The direction to be disabled is specified by the Most Significant Bit (MSB) of the endpointAndDirection parameter.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable
```

```
// a control endpoint. Note that the direction parameter is ignored.
// For a control endpoint, both the directions are disabled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBHS_DEVICE_EndpointDisable(handle, ep );

// This code shows an example of how to disable a BULK IN
// endpoint

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointDisable(handle, ep );
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointDisable
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBHS_DEVICE_EndpointDisableAll Function

This function disables all provisioned endpoints.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointDisableAll(DRV_HANDLE handle);
```

Returns

- USB_ERROR_NONE - The function exited successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function disables all provisioned endpoints in both directions.

Remarks

This function is typically called by the USB Device Layer to disable all endpoints upon detecting a bus reset.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable all endpoints.

DRV_USBHS_DEVICE_EndpointDisableAll(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointDisableAll(DRV_HANDLE handle)

DRV_USBHS_DEVICE_EndpointEnable Function

This function enables an endpoint for the specified direction and endpoint size.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointEnable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is not a valid endpoint defined for this driver instance. The value of DRV_USBHS_ENDPOINTS_NUMBER configuration constant should be adjusted.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function enables an endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, with the exception of control endpoints. If the endpoint type is a control endpoint, the endpoint is always bidirectional and the function needs to be called only once.

The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. A transfer that is scheduled over this endpoint will be scheduled in wMaxPacketSize transactions. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally reused.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to enable Endpoint
// 0 for control transfers. Note that for a control endpoint, the
// direction parameter is ignored. A control endpoint is always
// bidirectional. Endpoint size is 64 bytes.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);

// This code shows an example of how to set up a endpoint
// for BULK IN transfer. For an IN transfer, data moves from device
// to Host. In this example, Endpoint 1 is enabled. The maximum
// packet size is 64.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);

// If Endpoint 1 must also be set up for BULK OUT, the
// DRV_USBHS_DEVICE_EndpointEnable function must be called again, as shown
// here.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);
```

```
DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Function

```
USB_ERROR DRV_USBHS_DEVICE_EndpointEnable
(
    DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType,
uint16_t endpointSize
);
```

DRV_USBHS_DEVICE_EndpointIsEnabled Function

This function returns the enable/disable status of the specified endpoint and direction.

File

drv_usbhs.h

C

```
bool DRV_USBHS_DEVICE_EndpointIsEnabled(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- true - The endpoint is enabled.
- false - The endpoint is disabled.

Description

This function returns the enable/disable status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBHS_DEVICE_EndpointIsEnabled function can be used to obtain the
// status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(DRV_USBHS_ENDPOINT_STATE_DISABLED ==
            DRV_USBHS_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBHS_DEVICE_EndpointIsEnabled
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBHS_DEVICE_EndpointIsStalled Function

This function returns the stall status of the specified endpoint and direction.

File

drv_usbhs.h

C

```
bool DRV_USBHS_DEVICE_EndpointIsStalled(DRV_HANDLE client, USB_ENDPOINT endpoint);
```

Returns

- true - The endpoint is stalled.
- false - The endpoint is not stalled.

Description

This function returns the stall status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBHS_DEVICE_EndpointIsStalled function can be used to obtain the
// stall status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(true == DRV_USBHS_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    DRV_USBHS_DEVICE_EndpointStallClear(handle, ep);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBHS_DEVICE_EndpointIsStalled
(

DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBHS_DEVICE_EndpointStall Function

This function stalls an endpoint in the specified direction.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointStall(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_OSAL_FUNCTION - An error with an OSAL function called in this function.

Description

This function stalls an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to stall an endpoint. In
// this example, Endpoint 1 IN direction is stalled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointStall(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointStall
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBHS_DEVICE_EndpointStallClear Function

This function clears the stall on an endpoint in the specified direction.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointStallClear(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Description

This function clears the stall on an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to clear a stall. In this
// example, the stall condition on Endpoint 1 IN direction is cleared.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointStallClear(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointStallClear

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBHS_DEVICE_IRPCancel Function

This function cancels the specific IRP that are queued and in progress at the specified endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPCancel(DRV_HANDLE client, USB_DEVICE_IRP * irp);
```

Returns

- USB_ERROR_NONE - The IRP have been canceled successfully.
- USB_ERROR_PARAMETER_INVALID - Invalid parameter or the IRP already has been aborted or completed
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function attempts to cancel the processing of a queued IRP. An IRP that was in the queue but yet to be processed will be cancelled successfully and the IRP callback function will be called from this function with the USB_DEVICE_IRP_STATUS_ABORTED status. The application can release the data buffer memory used by the IRP when this callback occurs. If the IRP was in progress (a transaction in on the bus) when the cancel function was called, the IRP will be canceled only when an ongoing or the next transaction has completed. The IRP callback

function will then be called in an interrupt context. The application should not release the related data buffer unless the IRP callback has occurred.

Remarks

The size returned after the ABORT callback will be always 0 regardless of the amount of data that has been sent or received. The client should not assume any data transaction has happened for an canceled IRP. If the last transaction of the IRP was in progress, the IRP cancel does not have any effect. The first transaction of any ongoing IRP cannot be canceled.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel IRP.  In this example the IRP
// has been scheduled from a device to the Host.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBHS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // Check the status of the IRP.
    if(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Cancel the submitted IRP.
        if (DRV_USBHS_DEVICE_IRPCancel(handle, &irp) != USB_ERROR_NONE)
        {
            // The IRP Cancel request submission was successful.
            // IRP cancel status will be notified through the callback
            // function.
        }
        else
        {
            // The IRP may have been completed before IRP cancel operation.
            // could start. No callback notification will be generated.
        }
    }
    else
    {
        // The IRP processing must have been completed before IRP cancel was
        // submitted.
    }
}

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if the IRP callback is for a Cancel request
    if(irp->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        // IRP cancel completed
    }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
irp	Pointer to the IRP to cancel.

Function

USB_ERROR DRV_USBHS_DEVICE_IRPCancel
(
DRV_HANDLE client,
USB_DEVICE_IRP * irp
)

DRV_USBHS_DEVICE_IRPCancelAll Function

This function cancels all IRPs that are queued and in progress at the specified endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPCancelAll(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel all IRPs.

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if this is setup command

    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)
    {
        if(IsSetupCommandSupported(irp->data) == false)
        {
            // This means that this setup command is not
            // supported. Stall the some related endpoint and cancel all
            // queue IRPs.

            DRV_USBHS_DEVICE_EndpointStall(handle, ep);
            DRV_USBHS_DEVICE_IRPCancelAll(handle, ep);
        }
     }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBHS_DEVICE_IRPCancelAll

(
DRV_HANDLE client,
USB_ENDPOINT endpointAndDirection
);

DRV_USBHS_DEVICE_IRPSubmit Function

This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPSubmit(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection, USB_DEVICE_IRP *
irp);
```

Returns

- USB_ERROR_NONE - if the IRP was submitted successful.
- USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct.
- USB_ERROR_PARAMETER_INVALID - If the client handle is not valid.
- USB_ERROR_ENDPOINT_NOT_CONFIGURED - If the endpoint is not enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The specified endpoint is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL call in the function did not complete successfully.

Description

This function submits an I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to send and receive data from the USB Host. The data will be sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection parameter. Submitting an IRP arms the endpoint to either send data to or receive data from the Host. If an IRP is already being processed on the endpoint, the subsequent IRP submit operation will be queued. The contents of the IRP (including the application buffers) should not be changed until the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the Host can be less than, greater than, equal to, or be an exact multiple of the maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter, while sending data to the Host is less than the maximum packet size, the transfer will complete in one transaction.
- If the size parameter, while sending data to the Host is greater than the maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter, while sending data to the Host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to send a Zero Length Packet(ZLP) by specifying the USB_DEVICE_IRP_FLAG_DATA_COMPLETE flag as the flag parameter.
- The size parameter, while receiving data from the Host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_ERROR_IRP_SIZE_INVALID result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the Host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the Host. The IRP status is returned as USB_DEVICE_IRP_STATUS_COMPLETED_SHORT.
- If a ZLP needs to be sent to Host, the IRP size should be specified as 0 and the flag parameter should be set as USB_DEVICE_IRP_FLAG_DATA_COMPLETE.
- If the IRP size is an exact multiple of the endpoint size, the client can request the driver to not send a ZLP by setting the flag parameter to USB_DEVICE_IRP_FLAG_DATA_PENDING. This flag indicates that there is more data pending in this transfer.
- Specifying a size less than the endpoint size along with the USB_DEVICE_IRP_FLAG_DATA_PENDING flag will cause the driver to return a USB_ERROR_IRP_SIZE_INVALID.
- If the size is greater than but not a multiple of the endpoint size, and the flag is specified as USB_DEVICE_IRP_FLAG_DATA_PENDING, the driver will send multiple of endpoint size number of bytes. For example, if the IRP size is 130 and the endpoint size if 64, the number of bytes sent will 128.

Remarks

This function can be called from the ISR of the USB module to associated with the client.

Preconditions

The Client handle should be valid.

Example

```
// The following code shows an example of how to schedule a IRP to send data
// from a device to the Host. Assume that the max packet size is 64 and
// and this data needs to sent over Endpoint 1. In this example, the
```

```
// transfer is processed as three transactions of 64, 64 and 2 bytes.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBHS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task function.
    }
}

// The following code shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
// endpoint size.

irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

// Note that while  receiving data from the Host, the size should be an
// exact multiple of the maximum packet size of the endpoint. In the
// following example, the DRV_USBHS_DEVICE_IRPSubmit function will return a
// USB_DEVICE_IRP_SIZE_INVALID value.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

irp.data = myDataBufferToSend;
irp.size = 60; // THIS SIZE IS NOT CORRECT
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the IRP to be added to the queue for processing.

Function

```
USB_ERROR DRV_USBHS_DEVICE_IRPSubmit
(
    DRV_HANDLE client,
USB_ENDPOINT endpointAndDirection,
USB_DEVICE_IRP * irp
);
```

DRV_USBHS_DEVICE_RemoteWakeupStart Function

This function causes the device to start Remote Wakeup Signalling on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to start Remote Wakeup Signalling on the bus. This function should be called when the device, presently placed in suspend mode by the Host, wants to be wakeup. Note that the device can do this only when the Host has enabled the device's Remote Wakeup capability.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling.

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBHS_DEVICE_RemoteWakeupStart(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);

DRV_USBHS_DEVICE_RemoteWakeupStop Function

This function causes the device to stop the Remote Wakeup Signalling on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to stop Remote Wakeup Signalling on the bus. This function should be called after the DRV_USBHS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Remarks

This function should be 1 to 15 milliseconds after the DRV_USBHS_DEVICE_RemoteWakeupStart function was called.

Preconditions

The handle should be valid. The DRV_USBHS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling. Wait for 10
// milliseconds and then stop the Remote Wakeup signaling

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBHS_DEVICE_RemoteWakeupStart(handle);
    DelayMilliSeconds(10);
    DRV_USBHS_DEVICE_RemoteWakeupStop(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);

DRV_USBHS_DEVICE_SOFNumberGet Function

This function will return the USB SOF packet number.

File

drv_usbhs.h

C

```
uint16_t DRV_USBHS_DEVICE_SOFNumberGet(DRV_HANDLE handle);
```

Returns

The SOF packet number.

Description

This function will return the USB SOF packet number..

Remarks

None.

Preconditions

This function will return a valid value only when the device is attached to the bus. The SOF packet count will not increment if the bus is suspended.

Example

```
// This code shows how the DRV_USBHS_DEVICE_SOFNumberGet function is called
// to read the current SOF number.

DRV_HANDLE handle;
uint16_t sofNumber;

sofNumber = DRV_USBHS_DEVICE_SOFNumberGet(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

uint16_t DRV_USBHS_DEVICE_SOFNumberGet(DRV_HANDLE handle);

DRV_USBHS_DEVICE_TestModeEnter Function

This function enables the specified USB 2.0 Test Mode.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_TestModeEnter(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The handle or the value of testMode parameter is not valid.

Description

This function causes the device to enter the specified USB 2.0 defined test mode. It is called in response to Set Feature command from the host. The wValue field of this command specifies the Test Mode to enter. The USB module will perform the action identified by the testMode parameter.

Remarks

This function should be called only when the USB device has attached to the Host at High speed and only in response to the Set Feature command from the Host.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// This code shows how the DRV_USBHS_DEVICE_TestModeEnter function is
// called to make the USB device enter the Test_J test mode.

DRV_USBHS_DEVICE_TestModeEnter(handle, USB_TEST_MODE_SELECTOR_TEST_J);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
testMode	This parameter identifies the USB 2.0 specification test mode (see table 9-7 of the USB 2.0 specification).

Function

USB_ERROR DRV_USBHS_DEVICE_TestModeEnter

(

DRV_HANDLE handle,

USB_TEST_MODE_SELECTORS testMode

);

DRV_USBHS_DEVICE_TestModeExit Function

This function disables the specified USB 2.0 Test Mode.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_TestModeExit(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The handle or the value of testMode parameter is not valid.

Description

This function causes the device to stop the specified USB 2.0 defined test mode. This function can be called after calling the DRV_USBHS_DEVICE_TestModeEnter function to stop the test mode execution.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// This code shows how the DRV_USBHS_DEVICE_TestModeEnter function is
// called to make the USB device enter the Test_J test mode.

DRV_USBHS_DEVICE_TestModeEnter(handle, USB_TEST_MODE_SELECTOR_TEST_J);

// Now the DRV_USBHS_DEVICE_TestModeExit function is called to stop the
// Test_J test mode.

DRV_USBHS_DEVICE_TestModeExit(handle, USB_TEST_MODE_SELECTOR_TEST_J);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
testMode	This parameter identifies the USB 2.0 specification test mode (see table 9-7 of the USB 2.0 specification).

Function

USB_ERROR DRV_USBHS_DEVICE_TestModeExit
(
    DRV_HANDLE handle,
USB_TEST_MODE_SELECTORS testMode
);

d) Host Mode Operation Functions

DRV_USBHS_HOST_EventsDisable Function

Disables Host mode events.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_EventsDisable(DRV_HANDLE handle);
```

Returns

- true - Driver event generation was enabled when this function was called.
- false - Driver event generation was not enabled when this function was called.

Description

This function disables the Host mode events. This function is called by the Host Layer when it wants to execute code atomically.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_EventsDisable and
// DRV_USBHS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBHS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBHS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

bool DRV_USBHS_HOST_EventsDisable

(

DRV_HANDLE handle

);

DRV_USBHS_HOST_EventsEnable Function

Restores the events to the specified the original value.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_EventsEnable(DRV_HANDLE handle, bool eventContext);
```

Returns

None.

Description

This function will restore the enable disable state of the events. The eventRestoreContext parameter should be equal to the value returned by the DRV_USBHS_HOST_EventsDisable function.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_EventsDisable and
// DRV_USBHS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
```

```
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBHS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBHS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
eventRestoreContext	Value returned by the DRV_USBHS_HOST_EventsDisable function.

Function

void DRV_USBHS_HOST_EventsEnable

(

DRV_HANDLE handle

bool eventRestoreContext

);

DRV_USBHS_HOST_IRPCancel Function

Cancels the specified IRP.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);
```

Returns

None.

Description

This function attempts to cancel the specified IRP. If the IRP is queued and its processing has not started, it will be cancelled successfully. If the IRP in progress, the ongoing transaction will be allowed to complete.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how a submitted IRP can be cancelled.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

DRV_USBHS_HOST_IRPSubmit(controlPipeHandle, &irp);
```

```
// Additional application logic may come here. This logic may decide to
// cancel the submitted IRP.

DRV_USBHS_HOST_IRPCancel(&irp);
```

Parameters

Parameters	Description
inputIRP	Pointer to the IRP to cancel.

Function

void DRV_USBHS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);

DRV_USBHS_HOST_IRPSubmit Function

Submits an IRP on a pipe.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_IRPSubmit(DRV_USBHS_HOST_PIPE_HANDLE hPipe, USB_HOST_IRP * pinputIRP);
```

Returns

- USB_ERROR_NONE - The IRP was submitted successfully.
- USB_ERROR_PARAMETER_INVALID - The pipe handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An error occurred in an OSAL function called in this function.

Description

This function submits an IRP on the specified pipe. The IRP will be added to the queue and will be processed in turn. The data will be transferred on the bus based on the USB bus scheduling rules. When the IRP has been processed, the callback function specified in the IRP will be called. The IRP status will be updated to reflect the completion status of the IRP.

Remarks

An IRP can also be submitted in an IRP callback function.

Preconditions

The pipe handle should be valid.

Example

```
// The following code shows an example of how the host layer populates
// the IRP object and then submits it. IRP_Callback function is called when an
// IRP has completed processing. The status of the IRP at completion can be
// checked in the status flag. The size field of the irp will contain the amount
// of data transferred.

void IRP_Callback(USB_HOST_IRP * irp)
{
    // irp is pointing to the IRP for which the callback has occurred. In most
    // cases this function will execute in an interrupt context. The application
    // should not perform any hardware access or interrupt un-safe operations in
    // this function.

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            // IRP was terminated due to an unknown error
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            // IRP was terminated by the application
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            // IRP was terminated due to a bus error
```

```
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            // IRP was terminated due to data error
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            // IRP was terminated because of a NAK timeout
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            // IRP was terminated because of a device sent a STALL
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            // IRP has been completed
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            // IRP has been completed but the amount of data processed was less
            // than requested.
            break;

        default:
            break;
    }
}

// In the following code snippet the a control transfer IRP is submitted to a
// control pipe. The setup parameter of the IRP points to the Setup command of
// the control transfer. The direction of the data stage is specified by the
// Setup packet.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBHS_HOST_IRPSubmit(controlPipeHandle, &irp);
```

Parameters

Parameters	Description
hPipe	Handle to the pipe to which the IRP has to be submitted.
pInputIRP	Pointer to the IRP.

Function

```
USB_ERROR DRV_USBHS_HOST_IRPSubmit
(
    DRV_USBHS_HOST_PIPE_HANDLE  hPipe,
USB_HOST_IRP * pInputIRP
);
```

DRV_USBHS_HOST_PipeClose Function

Closes an open pipe.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_PipeClose(DRV_USBHS_HOST_PIPE_HANDLE pipeHandle);
```

Returns

None.

Description

This function closes an open pipe. Any IRPs scheduled on the pipe will be aborted and IRP callback functions will be called with the status as DRV_USB_HOST_IRP_STATE_ABORTED. The pipe handle will become invalid and the pipe and will not accept IRPs.

Remarks

None.

Preconditions

The pipe handle should be valid.

Example

```
// This code shows how an open Host pipe can be closed.

DRV_HANDLE driverHandle;
DRV_USBHS_HOST_PIPE_HANDLE pipeHandle;

// Close the pipe.
DRV_USBHS_HOST_PipeClose(pipeHandle);
```

Parameters

Parameters	Description
pipeHandle	Handle to the pipe to close.

Function

void DRV_USBHS_HOST_PipeClose

(

DRV_USBHS_HOST_PIPE_HANDLE pipeHandle

);

DRV_USBHS_HOST_PipeSetup Function

Open a pipe with the specified attributes.

File

drv_usbhs.h

C

```
DRV_USBHS_HOST_PIPE_HANDLE DRV_USBHS_HOST_PipeSetup(DRV_HANDLE client, uint8_t deviceAddress, USB_ENDPOINT endpointAndDirection, uint8_t hubAddress, uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval, uint16_t wMaxPacketSize, USB_SPEED speed);
```

Returns

- DRV_USB_HOST_PIPE_HANDLE_INVALID - The pipe could not be created.
- A valid Pipe Handle - The pipe was created successfully. This is an arbitrary value and will never be the same as DRV_USB_HOST_PIPE_HANDLE_INVALID.

Description

This function opens a communication pipe between the Host and the device endpoint. The transfer type and other attributes are specified through the function parameters. The driver does not check for available bus bandwidth, which should be done by the application (the USB Host Layer in this case)

Remarks

None.

Preconditions

The driver handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_PipeSetup function is called for
// create a communication pipe. In this example, Bulk pipe is created
// between the Host and a device. The Device address is 2 and the target
// endpoint on this device is 4 . The direction of the data transfer over
// this pipe is from the Host to the device. The device is connected to Port
// 1 of a Hub, whose USB address is 3. The maximum size of a transaction
// on this pipe is 64 bytes. This is a Bulk Pipe and hence the bInterval
// field is set to 0. The target device is operating at Full Speed.

DRV_HANDLE driverHandle;
DRV_USBHS_HOST_PIPE_HANDLE pipeHandle;

pipeHandle = DRV_USBHS_HOST_PipeSetup(driverHandle, 0x02, 0x14, 0x03, 0x01, USB_TRANSFER_TYPE_BULK, 0, 64,
USB_SPEED_FULL);

if(pipeHandle != DRV_USBHS_HOST_PIPE_HANDLE_INVALID)
{
    // The pipe was created successfully.
}
```

Parameters

Parameters	Description
client	Handle to the driver (returned from DRV_USBHS_Open function).
deviceAddress	USB Address of the device to connect to.
endpoint	Endpoint on the device to connect to.
hubAddress	Address of the hub to which this device is connected. If not connected to a hub, this value should be set to 0.
hubPort	Port number of the hub to which this device is connected.
pipeType	Transfer type of the pipe to open.
bInterval	Polling interval for periodic transfers. This should be specified as defined by the USB 2.0 Specification.
wMaxPacketSize	This should be set to the endpoint size reported by the device in its configuration descriptors. This defines the maximum size of the transaction in a transfer on this pipe.
speed	The speed of the pipe. This should match the speed at which the device connected to the Host.

Function

```
  DRV_USBHS_HOST_PIPE_HANDLE DRV_USBHS_HOST_PipeSetup
(
    DRV_HANDLE client,
uint8_t deviceAddress,
USB_ENDPOINT endpointAndDirection,
uint8_t hubAddress,
uint8_t hubPort,
USB_TRANSFER_TYPE pipeType,
uint8_t bInterval,
uint16_t wMaxPacketSize,
USB_SPEED speed
);
```

e) Root Hub Functions

DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet Function

This function returns the operating speed of the bus to which this root hub is connected.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_HIGH - The Root hub is connected to a bus that is operating at High Speed.
- USB_SPEED_FULL - The Root hub is connected to a bus that is operating at Full Speed.

Description

This function returns the operating speed of the bus to which this root hub is connected.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet function is
// called to know the operating speed of the bus to which this Root hub is
// connected.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

USB_SPEED DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_Initialize Function

This function initializes the root hub driver.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_ROOT_HUB_Initialize(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

Returns

None.

Description

This function initializes the root hub driver. It is called by the Host Layer at the time of processing the root hub devices. The Host Layer assigns a USB_HOST_DEVICE_INFO reference to this root hub driver. This identifies the relationship between the root hub and the Host Layer.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the USB Host Layer calls the
// DRV_USBHS_HOST_ROOT_HUB_Initialize function. The usbHostDeviceInfo
// parameter is an arbitrary identifier assigned by the USB Host Layer. Its
// interpretation is opaque to the Root hub Driver.

DRV_HANDLE drvHandle;
USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo = 0x10003000;

DRV_USBHS_HOST_ROOT_HUB_Initialize(drvHandle, usbHostDeviceInfo);
```

Parameters

Parameters	Description
handle	Handle to the driver.
usbHostDeviceInfo	Reference provided by the Host.

Function

void DRV_USBHS_HOST_ROOT_HUB_Initialize

(

DRV_HANDLE handle,

USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo,

)

DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet Function

Returns the maximum amount of current that this root hub can provide on the bus.

File

drv_usbhs.h

C

```
uint32_t DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE handle);
```

Returns

Returns the maximum current (in milliamperes) that the root hub can supply.

Description

This function returns the maximum amount of current that this root hub can provide on the bus.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet
// function is called to obtain the maximum VBUS current that the Root hub
// can supply.

DRV_HANDLE driverHandle;
uint32_t currentMilliAmperes;

currentMilliAmperes = DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

uint32_t DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE);

DRV_USBHS_HOST_ROOT_HUB_OperationEnable Function

This function enables or disables root hub operation.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_ROOT_HUB_OperationEnable(DRV_HANDLE handle, bool enable);
```

Returns

None.

Description

This function enables or disables root hub operation. When enabled, the root hub will detect devices attached to the port and will request the Host Layer to enumerate the device. This function is called by the Host Layer when it is ready to receive enumeration requests from the Host. If the operation is disabled, the root hub will not detect attached devices.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBHS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBHS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
enable	If this is set to true, root hub operation is enabled. If this is set to false, root hub operation is disabled.

Function

void DRV_USBHS_HOST_ROOT_HUB_OperationEnable

(

DRV_HANDLE handle,

bool enable

);

DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled Function

Returns the operation enabled status of the root hub.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);
```

Returns

- true - Root hub operation is enabled.
- false - Root hub operation is not enabled.

Description

This function returns true if the DRV_USBHS_HOST_ROOT_HUB_OperationEnable function has completed enabling the Host.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBHS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBHS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

bool DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet Function

Returns the number of ports this root hub contains.

File

drv_usbhs.h

C

```
uint8_t DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);
```

Returns

This function will always return 1.

Description

This function returns the number of ports that this root hub contains.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet function can
// be called to obtain the number of Root hub ports.

DRV_HANDLE driverHandle;
uint8_t nPorts;

nPorts = DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

uint8_t DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_PortReset Function

Resets the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);
```

Returns

None.

Description

This function resets the root hub port. The reset duration is defined by DRV_USBHS_ROOT_HUB_RESET_DURATION. The status of the reset signaling can be checked using the DRV_USBHS_ROOT_HUB_PortResetIsComplete function.

Remarks

The root hub on the PIC32MZ USB controller contains only one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBHS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBHS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBHS_ROOT_HUB_PortResetIsComplete function should be called
```

```
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to reset.

Function

void DRV_USBHS_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);

DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete Function

Returns true if the root hub has completed the port reset operation.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete(DRV_HANDLE handle, uint8_t port);
```

Returns

- true - The reset signaling has completed.
- false - The reset signaling has not completed.

Description

This function returns true if the port reset operation has completed. It should be called after the DRV_USB_HOST_ROOT_HUB_PortReset function to check if the reset operation has completed.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBHS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBHS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBHS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to check

Function

bool DRV_USBHS_ROOT_HUB_PortResetIsComplete

(

DRV_HANDLE handle,

uint8_t port
);

DRV_USBHS_HOST_ROOT_HUB_PortResume Function

Resumes the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortResume(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function resumes the root hub. The resume duration is defined by DRV_USBHS_ROOT_HUB_RESUME_DURATION. The status of the resume signaling can be checked using the DRV_USBHS_ROOT_HUB_PortResumeIsComplete function.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortResume function is
// called to resume the specified port.

DRV_HANDLE driverHandle;

// Resume Port 0.
DRV_USBHS_HOST_ROOT_HUB_PortResume(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to resume.

Function

USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortResume
(
DRV_HANDLE handle,
uint8_t port
);

DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet Function

Returns the speed of at which the port is operating.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_SPEED_ERROR - This value is returned if the driver handle is not or if the speed information is not available or if the specified port is not valid.

- USB_SPEED_HIGH - A High Speed device has been connected to the port.
- USB_SPEED_FULL - A Full Speed device has been connected to the port.
- USB_SPEED_LOW - A Low Speed device has been connected to the port.

Description

This function returns the speed at which the port is operating.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet function is
// called to know the operating speed of the port. This also indicates the
// operating speed of the device connected to this port.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port number of the port to be analyzed..

Function

USB_SPEED DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet

(

DRV_HANDLE handle,

uint8_t port

);

DRV_USBHS_HOST_ROOT_HUB_PortSuspend Function

Suspends the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function suspends the root hub port.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortSuspend function is
// called to suspend the specified port.
```

```
DRV_HANDLE driverHandle;

// Suspend Port 0.
DRV_USBHS_HOST_ROOT_HUB_PortSuspend(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to suspend.

Function

USB_ERROR DRV_USBHS_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);

f) Data Types and Constants

DRV_USBHS_EVENT Enumeration

Identifies the different events that the Hi-Speed USB Driver provides.

File

drv_usbhs.h

C

```
typedef enum {
  DRV_USBHS_EVENT_ERROR = DRV_USB_EVENT_ERROR,
  DRV_USBHS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT,
  DRV_USBHS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT,
  DRV_USBHS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT,
  DRV_USBHS_EVENT_STALL = DRV_USB_EVENT_STALL,
  DRV_USBHS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT,
  DRV_USBHS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID,
  DRV_USBHS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID
} DRV_USBHS_EVENT;
```

Members

Members	Description
DRV_USBHS_EVENT_ERROR = DRV_USB_EVENT_ERROR	Bus error occurred and was reported
DRV_USBHS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT	Host has issued a device reset
DRV_USBHS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT	Resume detected while USB in suspend mode
DRV_USBHS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT	Idle detected
DRV_USBHS_EVENT_STALL = DRV_USB_EVENT_STALL	Stall handshake has occurred
DRV_USBHS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT	Device received SOF operation
DRV_USBHS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID	VBUS voltage had Session valid
DRV_USBHS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID	Session Invalid

Description

Hi-Speed USB Driver Events Enumeration.

This enumeration identifies the different events that are generated by the Hi-Speed USB Driver.

Remarks

None.

DRV_USBHS_EVENT_CALLBACK Type

Type of the Hi-Speed USB Driver event callback function.

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_EVENT_CALLBACK)(uintptr_t hClient, DRV_USBHS_EVENT eventType, void * eventData);
```

Returns

None.

Description

Type of the Hi-Speed USB Driver Event Callback Function.

Define the type of the Hi-Speed USB Driver event callback function. The client should register an event callback function of this type when it intends to receive events from the Hi-Speed USB Driver. The event callback function is registered using the DRV_USBHS_ClientEventCallBackSet function.

Remarks

None.

Parameters

Parameters	Description
hClient	Handle to the driver client that registered this callback function.
eventType	This parameter identifies the event that caused the callback function to be called.
eventData	Pointer to a data structure that is related to this event. This value will be NULL if the event has no related data.

DRV_USBHS_HOST_PIPE_HANDLE Type

Defines the Hi-Speed USB Driver Host Pipe Handle type.

File

drv_usbhs.h

C

```
typedef uintptr_t DRV_USBHS_HOST_PIPE_HANDLE;
```

Description

Hi-Speed USB Driver Host Pipe Handle.

This type definition defines the type of the Hi-Speed USB Driver Host Pipe Handle.

Remarks

None.

DRV_USBHS_INIT Structure

This type definition defines the Driver Initialization Data Structure.

File

drv_usbhs.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  USBHS_MODULE_ID usbID;
  bool stopInIdle;
  bool suspendInSleep;
  INT_SOURCE interruptSource;
  INT_SOURCE interruptSourceUSBDma;
  USB_SPEED operationSpeed;
```

```
  DRV_USBHS_OPMODES operationMode;
  void * endpointTable;
  uint32_t rootHubAvailableCurrent;
  DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;
  DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;
  DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;
} DRV_USBHS_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System Module Initialization
USBHS_MODULE_ID usbID;	Identifies the USB peripheral to be used. This should be the USB PLIB module instance identifier.
bool stopInIdle;	This should be set to true if the USB module must stop operation in Idle mode
bool suspendInSleep;	This should be set to true if the USB module must suspend when the CPU enters Sleep mode.
INT_SOURCE interruptSource;	Specify the interrupt source for the USB module. This should be the interrupt source for the USB module instance specified in usbID.
INT_SOURCE interruptSourceUSBDma;	Specify the interrupt source for the USB module specific DMA controller. This should be the USB DMA interrupt source for the USB Module instance specified in usbID.
USB_SPEED operationSpeed;	Specify the operational speed of the USB module. This should always be set to USB_SPEED_FULL.
DRV_USBHS_OPMODES operationMode;	Specify the operation mode of the USB module. This specifies if the USB module should operate as a Device, Host, or both (Dual Role operation).
void * endpointTable;	A pointer to the endpoint descriptor table. This should be aligned at 512 byte address boundary. The size of the table is equal to the DRV_USBHS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed in the application.
uint32_t rootHubAvailableCurrent;	Root hub available current in milliamperes. This specifies the amount of current that root hub can provide to the attached device. This should be specified in mA. This is required when the driver is required to operate in Host mode.
DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;	When operating in Host mode, the application can specify a Root hub port enable function. This parameter should point to Root hub port enable function. If this parameter is NULL, it implies that the port is always enabled.
DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;	When operating in Host mode, the application can specify a Root Port Indication. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Root Port Indication is not supported.
DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;	When operating is Host mode, the application can specify a Root Port Overcurrent detection. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Overcurrent detection is not supported.

Description

USB Device Driver Initialization Data.

This structure contains all the data necessary to initialize the Hi-Speed USB Driver. A pointer to a structure of this type, containing the desired initialization data, must be passed into the DRV_USBHS_Initialize function.

Remarks

None.

DRV_USBHS_OPMODES Enumeration

Identifies the operating modes supported by the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
typedef enum {
  DRV_USBHS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE,
  DRV_USBHS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE,
  DRV_USBHS_OPMODE_HOST = DRV_USB_OPMODE_HOST,
  DRV_USBHS_OPMODE_OTG = DRV_USB_OPMODE_OTG
} DRV_USBHS_OPMODES;
```

Members

Members	Description
DRV_USBHS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE	The driver should be able to switch between Host and Device mode
DRV_USBHS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE	The driver should support Device mode operation only
DRV_USBHS_OPMODE_HOST = DRV_USB_OPMODE_HOST	The driver should support Host mode operation only
DRV_USBHS_OPMODE_OTG = DRV_USB_OPMODE_OTG	The driver should support the OTG protocol

Description

USB Operating Modes Enumeration.

This enumeration identifies the operating modes supported by the Hi-Speed USB Driver.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_INDICATION Type

USB Root hub Application Hooks (Port Indication).

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_ROOT_HUB_PORT_INDICATION)(uint8_t port, USB_HUB_PORT_INDICATOR_COLOR color,
USB_HUB_PORT_INDICATOR_STATE state);
```

Description

USB Root hub Application Hooks (Port Indication).

A function of the type defined here should be provided to the driver root to implement Port Indication. The root hub driver calls this function when it needs to update the state of the port indication LEDs. The application can choose to implement the Amber and Green colors as one LED or two different LEDs. The root hub driver specifies the color and the indicator attribute (on, off or blinking) when it calls this function.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT Type

USB Root hub Application Hooks (Port Overcurrent detection).

File

drv_usbhs.h

C

```
typedef bool (* DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT)(uint8_t port);
```

Description

USB Root hub Application Hooks (Port Overcurrent detection).

A function of the type defined here should be provided to the driver root hub to check for port over current condition. This function will be called periodically by the root hub driver to check the Overcurrent status of the port. It should continue to return true while the Overcurrent condition exists on the port. It should return false when the Overcurrent condition does not exist.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE Type

USB Root hub Application Hooks (Port Power Enable/ Disable).

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE)(uint8_t port, bool control);
```

Description

USB Root hub Application Hooks (Port Power Enable/ Disable).

A function of the type defined here should be provided to the driver root to control port power. The root hub driver will call this function when it needs to enable port power. If the application circuit contains a VBUS switch, the switch should be accessed and controlled by this function. If the enable parameter is true, the switch should be enabled and VBUS should be available on the port. If the enable parameter is false, the switch should be disabled and VBUS should not be available on the port.

Remarks

None.

DRV_USBHS_DEVICE_INTERFACE Macro

Hi-Speed USB Driver Device Mode Interface Functions.

File

drv_usbhs.h

C

```
#define DRV_USBHS_DEVICE_INTERFACE
```

Description

Hi-Speed USB Driver Device Mode Interface Functions.

The Device Controller Driver Interface member of the Device Stack Initialization data structure should be set to this value so that the Device Stack can access the Hi-Speed USB Driver Device Mode functions.

Remarks

None.

DRV_USBHS_HOST_INTERFACE Macro

Hi-Speed USB Driver Host Mode Interface Functions.

File

drv_usbhs.h

C

```
#define DRV_USBHS_HOST_INTERFACE
```

Description

Hi-Speed USB Driver Host Mode Interface Functions.

The Host Controller Driver Interface member of the Host Layer Initialization data structure should be set to this value so that the Host Layer can access the Hi-Speed USB Driver Host Mode functions.

Remarks

None.

DRV_USBHS_HOST_PIPE_HANDLE_INVALID Macro

Value of an Invalid Host Pipe Handle.

File

drv_usbhs.h

C

```
#define DRV_USBHS_HOST_PIPE_HANDLE_INVALID
```

Description

Hi-Speed USB Driver Invalid Host Pipe Handle.

This constant defines the value of an Invalid Host Pipe Handle.

Remarks

None.

DRV_USBHS_INDEX_0 Macro

Hi-Speed USB Driver Module Index 0 Definition.

File

drv_usbhs.h

C

```
#define DRV_USBHS_INDEX_0 0
```

Description

Hi-Speed USB Driver Module Index 0 Definition.

This constant defines the value of Hi-Speed USB Driver Index 0. The SYS_MODULE_INDEX parameter of the DRV_USBHS_Initialize and DRV_USBHS_Open functions should be set to this value to identify instance 0 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBHS_Initialize and DRV_USBHS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

Files

Files

Name	Description
drv_usbhs.h	PIC32MZ USB Module Driver Interface File
drv_usbhs_config_template.h	Hi-Speed USB (USBHS) Driver Configuration Template.

Description

drv_usbhs.h

PIC32MZ USB Module Driver Interface File

Enumerations

	Name	Description
	DRV_USBHS_EVENT	Identifies the different events that the Hi-Speed USB Driver provides.
	DRV_USBHS_OPMODES	Identifies the operating modes supported by the Hi-Speed USB Driver.

Functions

	Name	Description
	DRV_USBHS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBHS_Close	Closes an opened-instance of the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBHS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBHS_DEVICE_CurrentSpeedGet	This function will return the USB speed at which the device is operating.

	DRV_USBHS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBHS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBHS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBHS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBHS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBHS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBHS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.
	DRV_USBHS_DEVICE_TestModeEnter	This function enables the specified USB 2.0 Test Mode.
	DRV_USBHS_DEVICE_TestModeExit	This function disables the specified USB 2.0 Test Mode.
	DRV_USBHS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBHS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBHS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBHS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBHS_HOST_PipeClose	Closes an open pipe.
	DRV_USBHS_HOST_PipeSetup	Open a pipe with the specified attributes.
	DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBHS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBHS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBHS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBHS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBHS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.
	DRV_USBHS_Initialize	Initializes the Hi-Speed USB Driver.
	DRV_USBHS_Open	Opens the specified Hi-Speed USB Driver instance and returns a handle to it.
	DRV_USBHS_Status	Provides the current status of the Hi-Speed USB Driver module.
	DRV_USBHS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBHS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.
	DRV_USBHS_Tasks_ISR_USBDMA	Maintains the driver's DMA Transfer state machine and implements its ISR.

Macros

	Name	Description
	DRV_USBHS_DEVICE_INTERFACE	Hi-Speed USB Driver Device Mode Interface Functions.
	DRV_USBHS_HOST_INTERFACE	Hi-Speed USB Driver Host Mode Interface Functions.

	DRV_USBHS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBHS_INDEX_0	Hi-Speed USB Driver Module Index 0 Definition.

Structures

	Name	Description
	DRV_USBHS_INIT	This type definition defines the Driver Initialization Data Structure.

Types

	Name	Description
	DRV_USBHS_EVENT_CALLBACK	Type of the Hi-Speed USB Driver event callback function.
	DRV_USBHS_HOST_PIPE_HANDLE	Defines the Hi-Speed USB Driver Host Pipe Handle type.
	DRV_USBHS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).

Description

PIC32MZ USB Module Driver Interface Header File

The PIC32MZ Hi-Speed USB Module driver provides a simple interface to manage the "USB" peripheral on the PIC32MZ microcontroller. This file defines the interface definitions and prototypes for the Hi-Speed USB Driver. The driver interface meets the requirements of the MPLAB Harmony USB Host and Device Layer.

File Name

drv_usbhs.h

Company

Microchip Technology Inc.

drv_usbhs_config_template.h

Hi-Speed USB (USBHS) Driver Configuration Template.

Macros

	Name	Description
	DRV_USBHS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBHS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.
	DRV_USBHS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBHS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBHS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBHS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBHS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBHS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

Hi-Speed USB Driver Configuration Template

This file lists all the configurations constants that affect the operation of the USBHS Driver.

File Name

drv_usbhs_config_template.h

Company

Microchip Technology Inc.

USART Driver Library

This section describes the USART Driver Library.

Introduction

This section introduces the MPLAB Harmony USART Driver.

Description

The MPLAB Harmony USART Driver (also referred to as the USART Driver) provides a high-level interface to the USART and UART peripherals on Microchip's PIC32 microcontrollers. This driver provides application ready routines to read and write data to the UART using common data transfer models, which eliminates the need for the application to implement this code. The USART driver features the following:

- Provides byte transfer, read/write, and buffer queue data transfer models
- Supports Interrupt and Polled modes of operation
- Supports point-to-point data communication
- Supports multi-client and multi-instance operation
- Provides data transfer events
- Supports blocking and non-blocking operation with the read/write data transfer model
- Features thread-safe functions for use in RTOS applications
- Supports DMA transfers
- Supports high baud rate setting
- Major features are implemented in separate source code files and can be included only if needed. This helps optimize overall application code size.

Using the Library

This topic describes the basic architecture of the USART Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usart.h`

The interface to the USART library is defined in the `drv_usart.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

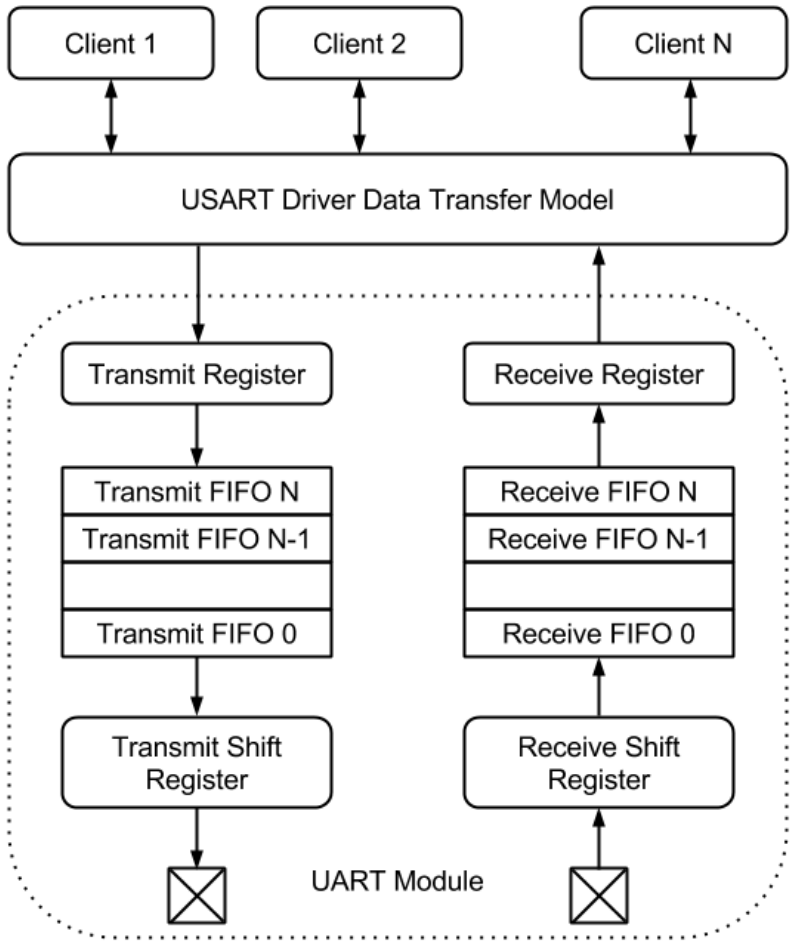
This section describes how the USART Driver abstracts the USART peripheral features.

Description

The USART driver features routines to perform the following functions:

- Driver initialization
- Transfer data
- Manage communication properties of the module

The Driver initialization routines allow the system to initialize the driver. The driver must be initialized before it can be opened by a client. The data transfer routines allow the application to receive and transmit data through the USART. The driver also provides routines to change the communication properties such as USART baud or line control settings.



As seen in the previous figure, the USART driver clients transfer data through the USART Driver Data Transfer model. The driver abstracts out the hardware details of the UART module FIFO mechanism and shift registers, and provides a low overhead data transfer mechanism to the application. The USART driver provides three different data transfer models for transferring data.

- The Byte Transfer Model
- The File I/O Type Read/Write Transfer Model
- Buffer Queue Transfer Model

Byte Transfer Model:

The byte transfer model allows the application to transfer data through USART driver one byte at a time. With this model, the driver reads one byte from the receive FIFO or writes one byte to the transmit FIFO. The application must check if data has been received before reading the data. Similarly, it must check if the transmit FIFO is not full before writing to the FIFO. The byte transfer model places the responsibility of maintaining the USART peripheral on the Application. The driver cannot support other data transfer models if support for this data transfer model is enabled. The byte transfer model is only recommended for simple data transfer applications.

To use the byte transfer model, the `drv_usart_byte_model.c` file must be included in the project and the DRV_USART_BYTE_MODEL_SUPPORT configuration macro should be set to true.

File I/O Type Read/Write Transfer Model:

This data transfer model is similar to file read and write API model in a UNIX operating system application. The application calls the USART driver read and write routines to transfer data through the USART. Unlike the byte transfer model, the read/write data model can process a block of data. Depending on the mode (blocking or non-blocking) in which the client opened the driver, the driver will either block until all of the data is transferred or will immediately return with the number of bytes transferred. The application does not have to check the FIFO status while using this mode. The application can instead use the return status (number of bytes transferred) to maintain its logic and complete the data transfer. The read/write model can be used with the non-DMA buffer queue model. It cannot be used with the byte transfer model and the DMA-enabled buffer queue model in the same application.

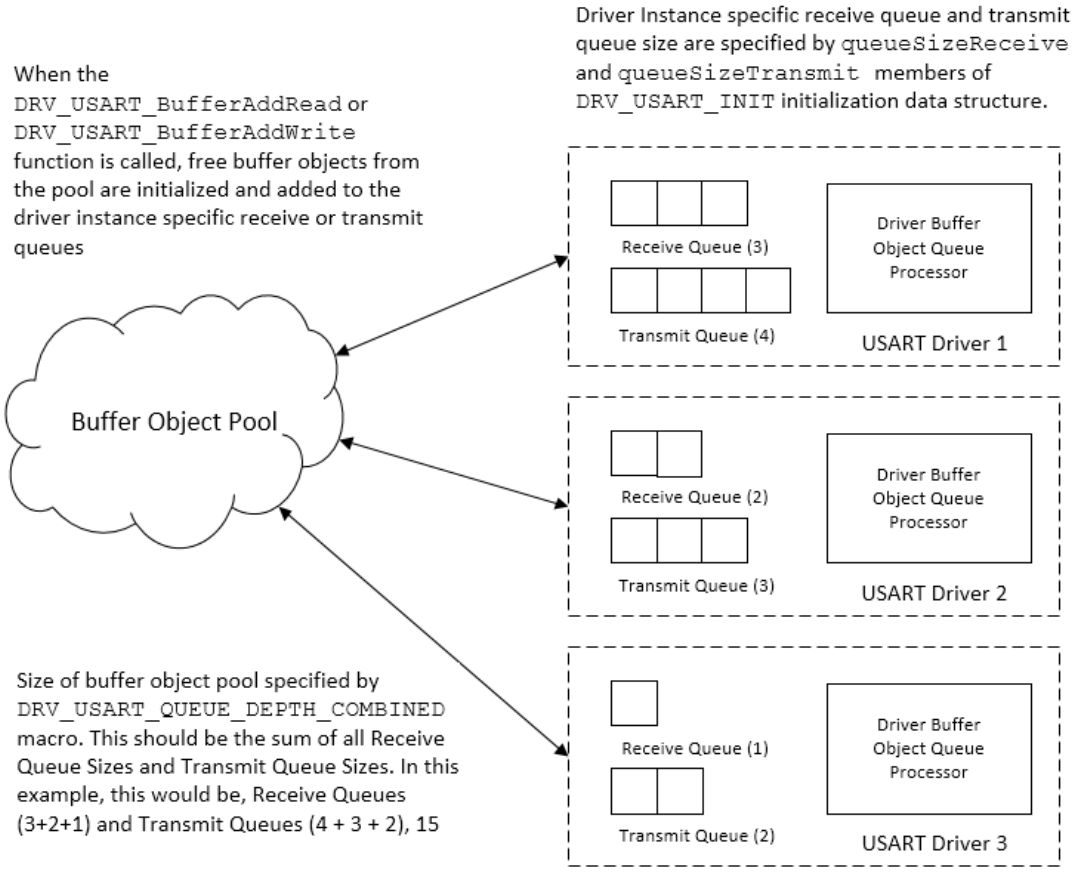
To use the file I/O type read/write data transfer model, the `drv_usart_read_write.c` file must be included in the project and the DRV_USART_READ_WRITE_MODEL_SUPPORT configuration macro should be set to true.

See File I/O Type Read/Write Data Transfer Modef for additional information.

Buffer Queue Transfer Model:

The buffer queue data transfer model allows clients to queue data transfers for processing. This data transfer model is always non-blocking. The USART driver returns a buffer handle for a queued request. The clients can track the completion of a buffer through events and API. If the USART driver is busy processing a data transfer, other data transfer requests are queued. This allows the clients to optimize their application logic and increase throughput. To optimize memory usage, the USART driver implements a shared buffer object pool concept to add a data transfer request to the queue. The following figure shows a conceptual representation of the buffer queue model.

Buffer Queue Transfer Model



As shown in the previous figure, each USART driver hardware instance has a read and write queue. The system designer must configure the sizes of these read and write queues. The USART driver additionally employs a global pool of buffer queue objects. This pool is common to all USART Driver hardware instances and its size is defined by the DRV_USART_QUEUE_DEPTH_COMBINED configuration macro. When a client places a request to add a data transfer, the driver performs the following actions:

- It checks if a buffer object is free in the global pool. If not, the driver rejects the request.
- It then checks if the hardware instance specific queue is full. If not, the buffer object from the global pool is added to the hardware instance specific queue. If the queue is full, the driver rejects the request.

The buffer queue model can be used along with the file I/O type read/write data transfer model.

To use the Buffer Queue Data Transfer model, the `drv_usart_buffer_queue.c` file must be included in the project and DRV_USART_BUFFER_QUEUE_SUPPORT configuration macro should be set to true.

The USART Driver DMA feature is only available while using the Buffer Queue Model. If enabled, the USART Driver uses the DMA module channels to transfer data directly from application memory to USART transmit or receive registers. This reduces CPU resource consumption and improves system performance. To use the buffer queue model with DMA, the `drv_usart_buffer_queue_dma.c` file should be included in the project instead of `drv_usart_buffer_queue.c`.

See Buffer Queue Transfer Model for additional information.

Communication Management

The USART Driver API contains functions to control the USART Driver communication properties. These functions allow the client to change the parity, stop bits, number of data bits and the communication baud rate. A change in the communication setting affects all ongoing communication and all driver clients.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART Driver Library.

Library Interface Section	Description
System Routines	These routines are accessed by the MPLAB Harmony system module. They allow the driver to be initialized, deinitialized and maintained.
Core Client Routines	These routines allow the application client to open and close the driver.
Communication Management Client Routines	These routines allow the client to change the properties of the communication channel (such as baud, parity, etc.).
Buffer Queue Read/Write Client Routines	These routines allow the client to use the buffer queue data transfer model.
File I/O Type Read/Write Routines	These routines allow the client to use the file I/O type read/write routines.
Byte Transfer Routines	These routines allow the client to use the byte data transfer model.

The USART driver must be first initialized. One or more application clients can then open the USART Driver in Blocking or non-Blocking mode. The Open function returns a handle which allows the client to access the driver client functionality. The Driver tasks routines should be invoked regularly from the SYS_Tasks routine in case of Polled mode operation or from USART Driver Interrupt Service Routine (ISR), in case of Interrupt mode.

The driver implementation is split across multiple files to optimize the application project code size. The application project must include the `drv_usart.c` file if the USART driver is needed in the application. If DMA-enabled data transfers are required, the `drv_usart_dma.c` file should be included into the project instead of the `drv_usart.c` file. These files implement the system and core Client routines. Other driver files can be included based on the required driver features.

The USART Driver API, unless otherwise specified, should not be called from an interrupt context. That is, they should not be called from an ISR or from event handlers that are executing within an ISR context.

How the Library Works

This section describes how to use the USART Driver.

Description

Prior to using the USART Driver, the application must decide on which USART data transfer models are required. The application project should then include the USART Driver files, required to support the data transfer model into the application project. Additionally, the application design must consider the need for USART Driver to be opened in blocking or non blocking modes. This will also affect the application flow.

Initializing the USART Driver

Describes how to initialize the USART Driver.

Description

The USART Driver must be configured and initialized for clients to be able to open the driver. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_USART_INIT data structure that is passed to the DRV_USART_Initialize function. The initialization parameters include the USART baud, the USART Peripheral, USART interrupts and read queue and write queue sizes (which are applicable only when buffer queue data transfer is used). The following code shows an example of initializing the USART driver for 300 bps and uses USART2. If the driver is configured for Interrupt mode of operation, the priority of the USART interrupts needs to be specified.

```
/* The following code shows an example of designing the
 * DRV_USART_INIT data structure. It also shows how an example
 * usage of the DRV_USART_Initialize() function and how Interrupt
 * System Service routines are used to set USART Interrupt
 * priority. */

DRV_USART_INIT usartInit;
SYS_MODULE_OBJ usartModule1;

/* Set the baud to 300 */
usartInit.baud = 300;

/* Auto Baud detection or Stop Idle is not needed */
usartInit.flags = DRV_USART_INIT_FLAG_NONE;

/* Handshaking is not needed */
usartInit.handshake = DRV_USART_HANDSHAKE_NONE;
```

```
/* USART Error Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_ERROR
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2*/
usartInit.interruptError = INT_SOURCE_USART_2_ERROR;

/* USART Receive Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_RECEIVE
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2 */
usartInit.interruptReceive = INT_SOURCE_USART_2_RECEIVE;

/* USART Transmit Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_TRANSMIT
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2 */
usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;

/* Line control mode */
usartInit.lineControl = DRV_USART_LINE_CONTROL_8NONE1;

/* Operation mode is normal. Loopback or addressed is not
 * needed */
usartInit.mode = DRV_USART_OPERATION_MODE_NORMAL;

/* Peripheral Bus clock frequency at which the USART is
 * operating */
usartInit.brgClock = 80000000;

/* System module power setting. Typically set to
 * SYS_MODULE_POWER_RUN_FULL */
usartInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;

/* Receive buffer queue size. In this case a maximum of 2
 * receive buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeReceive = 2;

/* Transmit buffer queue size. In this case a maximum of 3
 * transmit buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeTransmit = 3;

/* The USART peripheral instance index associated with this
 * driver instance. Note that this value is defined by the
 * USART Peripheral Library */
usartInit.usartID = USART_ID_2;

/* Initialize USART Driver Instance 0 */
usartModule1 = DRV_USART_Initialize(DRV_USART_0, (SYS_MODULE_INIT*)&usartInit);

/* The result of the driver initialization can be checked */
if(SYS_MODULE_OBJ_INVALID == usartModule1)
{
    /* There was an error in initialization. */
}

/* If the USART driver is configured for interrupt mode of
 * operation, the interrupt priorities should be configured.
 * Here the Interrupt System Service is used to set the
 * priority to level 4 */

/* Initialize the interrupt system service */
SYS_INT_Initialize();

/* Set the USART 2 module interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_UART2, INT_PRIORITY_LEVEL4);
```

```
    /* Set the USART 2 module interrupt sub-priority to 0*/
    SYS_INT_VectorSubprioritySet(INT_VECTOR_UART2, INT_SUBPRIORITY_LEVEL0);

    /* Enable global interrupt */
    SYS_INT_Enable();
```

The USART Driver can be configured to transfer data through the DMA. In such a case, the DMA channels to be used for USART transmit and receive need to be specified. The USART Driver depends on the DMA System Service to access the DMA module. The DMA channels to be used for transmit and receive transfers should be specified in the DRV_USART_INIT data structure.

The following code shows an example of using the USART Driver initialization to use DMA for transferring data. The code also shows example initialization of the DMA System Service.

```
  /* The following code shows an example of designing the
   * DRV_USART_INIT data structure. It also shows how an example
   * usage of the DRV_USART_Initialize() function and how Interrupt
   * System Service routines are used to set USART Interrupt
   * priority. */

    DRV_USART_INIT usartInit;
    SYS_DMA_INIT dmaInit;
    SYS_MODULE_OBJ usartModule1;
    SYS_MODULE_OBJ dmaModule;

    /* Set the baud to 300 */
    usartInit.baud = 300;

    /* Auto Baud detection or Stop Idle is not needed */
    usartInit.flags = DRV_USART_INIT_FLAG_NONE;

    /* Handshaking is not needed */
    usartInit.handshake = DRV_USART_HANDSHAKE_NONE;

    /* USART Error Interrupt source for this USART
     * driver instance. Note that INT_SOURCE_USART_2_ERROR
     * value is defined by the Interrupt System Service and
     * is the error interrupt for USART2*/
    usartInit.interruptError = INT_SOURCE_USART_2_ERROR;

    /* USART Receive Interrupt source for this USART
     * driver instance. Note that INT_SOURCE_USART_2_RECEIVE
     * value is defined by the Interrupt System Service and
     * is the receive interrupt for USART2 */
    usartInit.interruptReceive = INT_SOURCE_USART_2_RECEIVE;

    /* USART Transmit Interrupt source for this USART
     * driver instance. Note that INT_SOURCE_USART_2_TRANSMIT
     * value is defined by the Interrupt System Service and
     * is the transmit interrupt for USART2 */
    usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;

    /* Line control mode */
    usartInit.lineControl = DRV_USART_LINE_CONTROL_8NONE1;

    /* Operation mode is normal. Loopback or addressed is not
     * needed */
    usartInit.mode = DRV_USART_OPERATION_MODE_NORMAL;

    /* Peripheral Bus clock frequency at which the USART is
     * operating */
    usartInit.brgClock = 80000000;

    /* System module power setting. Typically set to
     * SYS_MODULE_POWER_RUN_FULL */
    usartInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;

    /* Receive buffer queue size. In this case a maximum of 2
     * receive buffers can be queued. Only applicable if the
     * Buffer Queue Data Transfer Model is included in the
     * application. */
```

```
usartInit.queueSizeReceive = 2;

/* Transmit buffer queue size. In this case a maximum of 3
 * transmit buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeTransmit = 3;

/* The USART peripheral instance index associated with this
 * driver instance. Note that this value is defined by the
 * USART Peripheral Library */
usartInit.usartID = USART_ID_2;

/* Use DMA channel 1 for transmit. If transmit via DMA is
 * not required, set this to DMA_CHANNEL_NONE. These values
 * are defined by the DMA System Service. */
usartInit.dmaChannelTransmit = DMA_CHANNEL_1;

/* Use DMA channel 2 for receive. If receive via DMA is
 * not required, set this to DMA_CHANNEL_NONE. These values
 * are defined by the DMA System Service. */
usartInit.dmaChannelReceive = DMA_CHANNEL_2;

/* Set the interrupt source for the Transmit DMA channel.
 * This parameter is ignored if the dmaChannelTransmit
 * parameter is set to DMA_CHANNEL_NONE. */
usartInit.dmaInterruptTransmit = INT_SOURCE_DMA_1;

/* Set the interrupt source for the Receive DMA channel.
 * This parameter is ignored if the dmaChannelReceive
 * parameter is set to DMA_CHANNEL_NONE. */
usartInit.dmaInterruptReceive = INT_SOURCE_DMA_2;

/********* End of DRV_USART_INIT Initialization *************/


/* If the USART driver is configured for interrupt mode of
 * operation, the interrupt priorities should be configured.
 * Here the Interrupt System Service is used to set the
 * priority to level 4 */

/* Initialize the interrupt system service */
SYS_INT_Initialize();

/* Set the USART 2 module interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_UART2, INT_PRIORITY_LEVEL4);

/* Set the USART 2 module interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_UART2, INT_SUBPRIORITY_LEVEL0);

/* Set the DMA 1 channel interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_DMA1, INT_PRIORITY_LEVEL4);

/* Set the DMA 1 channel interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_DMA1, INT_SUBPRIORITY_LEVEL0);

/* Set the DMA 2 channel interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_DMA2, INT_PRIORITY_LEVEL4);

/* Set the DMA 2 channel interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_DMA2, INT_SUBPRIORITY_LEVEL0);

/* Enable global interrupt */
SYS_INT_Enable();

/* This is the DMA System Service Initialization */
dmaInit.sidl = SYS_DMA_SIDL_DISABLE;
dmaModule = SYS_DMA_Initialize((SYS_MODULE_INIT*)&dmaInit);
```

```
    /* The result of the DMA System Service initialization can be checked */
    if(SYS_MODULE_OBJ_INVALID == dmaModule)
    {
        /* DMA System Service initialization was not successful */
    }

    /* Initialize USART Driver Instance 0 */
    usartModule1 = DRV_USART_Initialize(DRV_USART_0, (SYS_MODULE_INIT*)&usartInit);

    /* The result of the driver initialization can be checked */
    if(SYS_MODULE_OBJ_INVALID == usartModule1)
    {
        /* There was an error in initialization. */
    }
```

Opening the USART Driver

Describes how to open the USART Driver.

Description

To use the USART driver, the application must open the driver. This is done by calling the DRV_USART_Open function. Calling this function with DRV_IO_INTENT_NONBLOCKING will cause the driver to be opened in non blocking mode. The DRV_USART_Read and DRV_USART_Write functions when called by this client will be non blocking. . Calling this function with DRV_IO_INTENT_BLOCKING will cause the driver to be opened in blocking mode. The DRV_USART_Read and DRV_USART_Write functions when called by this client will be blocking.

If successful, the DRV_USART_Open function will return a valid handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_USART_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well.

The following code shows an example of the driver being opened in different modes.

```
DRV_HANDLE usartHandle1, usartHandle2;

/* Client 1 opens the USART driver in non blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* Client 2 opens the USART driver in blocking mode */
usartHandle2 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_BLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == usartHandle2)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* The client can also open the USART driver in read only mode
 * (DRV_IO_INTENT_READ), write only mode (DRV_IO_INTENT_WRITE)
 * and exclusive mode (DRV_IO_INTENT_EXCLUSIVE). If the driver
 * has been opened exclusively by a client, it cannot be opened
 * again by another client */
```

Byte Transfer Model

Describes the USART Driver byte transfer model.

Description

To use the byte transfer model, the DRV_USART_BYTE_MODEL_SUPPORT configuration macro should be true. The `drv_usart_byte_model.c` function should be included in the application project. The application cannot support the read/write and buffer queue data transfer model when the byte model is enabled.

The following code shows an example of how the DRV_USART_WriteByte function and the DRV_USART_ReadByte function are used.

```
 /* Client uses the a byte model API to write a byte*/
if(!DRV_USART_TransmitBufferIsFull(usartHandle1))
{
    byte = '1';
    DRV_USART_WriteByte(usartHandle1,byte);
}

/* Client waits until data is available and then reads
 * byte */
 while(DRV_USART_ReceiverBufferIsEmpty(usartHandle1));
        byte = DRV_USART_ReadByte(usartHandle1);
```

File I/O Type Read/Write Data Transfer Model

This topic describes the file I/O type read/write data transfer model.

Description

To use the file I/O type read/write data transfer model, the DRV_USART_READ_WRITE_MODEL_SUPPORT configuration macro should be 'true'. The file `drv_usart_read_write.c` file should be included in the application project. The driver can support the non-DMA buffer queue data transfer model along with the file I/O type read/write data transfer model. The byte transfer model and DMA buffer queue model cannot be enabled if the file I/O type read/write data transfer model is enabled.

The DRV_USART_Read and DRV_USART_Write function represent the file I/O type read/write data transfer model. The functional behavior of these API is affected by the mode in which the client opened the driver. If the client opened the driver in blocking mode these API will block. In blocking mode, the DRV_USART_Read and DRV_USART_Write functions will not return until the requested number of bytes have been read or written. When operating in a RTOS application, the application thread that has opened driver in blocking mode, will enter a blocked state when it calls DRV_USART_Write or DRV_USART_Read function. This will allow the RTOS scheduler to schedule other threads which are ready to run. If the client opened the driver in non-blocking mode these API will not block. In non-blocking mode, the DRV_USART_Read and DRV_USART_Write functions will return immediately with the amount of data that could be read or written.

Note: Do not open the driver in Blocking mode when the driver is configured for polling operation (DRV_USART_INTERRUPT_MODE is false) in a bare-metal (non-RTOS) application. This will cause the system to enter an infinite loop condition when the DRV_USART_Read or DRV_USART_Write function is called.

The following code shows an example of file I/O type read/write data transfer model usage when the driver is opened in Blocking mode.

```
/* This code shows the functionality of the DRV_USART_Write and
 * DRV_USART_Read function when the driver is opened in blocking mode */

DRV_HANDLE usartHandle1;
uint8_t myData[10];
size_t bytesProcessed;

/* The driver is opened in blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_BLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* Transmit 10 bytes from the myData array. Function will not return until 10 bytes
 * have been accepted by the driver. This is because the client opened the driver
 * in blocking mode. */

bytesProcessed = DRV_USART_Write(usartHandle1, myData, 10);

/* Read 10 bytes from the myData array. Function will not return until all 10 bytes
 * have been received by the driver. This is because the client opened the driver
 * in blocking mode. */

bytesProcessed = DRV_USART_Read(usartHandle1, myData, 10);
```

In non-Blocking mode, the driver uses the internal USART hardware FIFO as storage. The DRV_USART_Read function checks if bytes are available in USART receive hardware FIFO. If bytes are available, these are read and the number of bytes read is returned. The DRV_USART_Write function checks if USART transmit hardware FIFO has empty location. If locations are empty, the bytes to be transmitted are queued up in the FIFO and the number of queued bytes is returned. In either case, the number of bytes read or written may be less than the number requested by the client. The client can, in such a case, call the DRV_USART_Read and/or the DRV_USART_Write functions again to process the pending bytes. The following code shows how this can be done.

```
/* This code shows the functionality of the DRV_USART_Write and
 * DRV_USART_Read functions when the driver is opened in non-blocking mode */

DRV_HANDLE usartHandle1;
uint8_t myData[10];
size_t bytesProcessed;

/* The driver is opened in non-blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0,
        DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* The following code call the DRV_USART_Write function
 * multiple times to write 10 bytes completely. Note how the
 * function return value is used to update the location of
 * user source data. */

bytesProcessed = 0;
do
{
    /* Write data to the USART and use the return value to
     * update the source data pointer and pending bytes number. */
    bytesProcessed += DRV_USART_Write(usartHandle1,
            myData + bytesProcessed, (10 - bytesProcessed));

} while(bytesProcessed < 10);

/* The following code calls the DRV_USART_Read function multiple times to read
 * 10 bytes completely. Note how the function return value is used to update the
 * location of user destination array. */

bytesProcessed = 0;
do
{
    /* Read data from the USART and use the return value to update the
     * destination pointer and pending bytes number. */
    bytesProcessed += DRV_USART_Read(usartHandle1,
            myData + bytesProcessed, (10 - bytesProcessed));

}while (bytesProcessed < 10);
```

Buffer Queue Transfer Model

This topic describes the buffer queue data transfer model.

Description

To use the buffer queue data transfer model, the DRV_USART_BUFFER_QUEUE_SUPPORT configuration macro should be true. The file, `drv_usart_buffer_queue.c`, should be included in the application project. If the DMA-enabled buffer queue model is required, the `drv_usart_buffer_queue_dma.c` file (*and not* the `drv_usart_buffer_queue.c`) should be included in the application project. The DMA and non-DMA buffer queue model API is the same. The driver can support the non-DMA buffer queue data transfer model along with the file I/O type read/write data transfer model. The byte transfer model cannot be enabled if the buffer queue data transfer model is enabled.

The DRV_USART_BufferAddRead and DRV_USART_BufferAddWrite functions represent the buffer queue data transfer model. These functions are always non-blocking. The Buffer Queue Data Transfer Model employs queuing of read and write request. Each driver instance contains a read and write queue. The size of the read queue is determined by the queueSizeRead member of the DRV_USART_INIT data structure. The size of the write queue is determined by the queueSizeWrite member of the DRV_USART_INIT data structure. The driver provides driver events (DRV_USART_BUFFER_EVENT) that indicates termination of the buffer requests.

When the driver is configured for Interrupt mode operation (that is defined and registered by the driver client), the buffer event handler executes in an interrupt context. Calling computationally intensive or hardware polling routines within the event handlers is not recommended. Calling interrupt unsafe functions in the event handler when the driver is configured for Interrupt mode could result in unpredictable system behavior.

When the driver adds request to the queue, it returns a buffer handle. This unique handle allows the client to track the request as it progresses through the queue. The buffer handle is returned with the buffer event and expires when the event associated with the buffer has been generated and the event handler returns. The following code shows an example of using the buffer queue data transfer model.

```
/* This code shows an example of using the
 * Buffer Queue Data Transfer Model. */
DRV_HANDLE usartHandle1;
uint8_t myData1[10], myData2[10];
uint8_t myData3[10], myData4[10];
size_t bytesProcessed;
DRV_USART_BUFFER_HANDLE bufferHandle1, bufferHandle2;
DRV_USART_BUFFER_HANDLE bufferHandle3, bufferHandle4;


/* The driver is opened in non blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0,
        DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* Register a Buffer Event Handler with USART driver.
 * This event handler function will be called whenever
 * there is a buffer event.  An application defined
 * context can also be specified. This is returned when
 * the event handler is called.
 * */
DRV_USART_BufferEventHandlerSet(usartHandle1,
        APP_USARTBufferEventHandler, NULL);


/* Queue up two buffers for transmit */
DRV_USART_BufferAddWrite(usartHandle1, &bufferHandle1, myData1, 10);
DRV_USART_BufferAddWrite(usartHandle1, &bufferHandle2, myData2, 10);

/* Queue up two buffers for receive */
DRV_USART_BufferAddRead(usartHandle1, &bufferHandle3, myData3, 10);
DRV_USART_BufferAddRead(usartHandle1, &bufferHandle4, myData4, 10);

/* This is application USART Driver Buffer Event Handler */

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            /* This means the data was transferred */
            break;
        case DRV_USART_BUFFER_EVENT_ERROR:
            /* Error handling here. */
            break;
        default:
            break;
    }
}
```

Driver Tasks Routine

This topic describes the Driver "Task" routines.

Description

The USART driver contains three task routines, DRV_USART_TasksTransmit, DRV_USART_TasksReceive and DRV_USART_TasksError. These task routines implement the USART Driver state machines for transmit, receive and error related operations. If the driver is configured for polling operation, the required task routine should be called in SYS_Tasks routine of the system. If the driver is configured for interrupt mode of operation, the task routine should be called from the ISR. The following code shows an example of both.

```
/* The following code shows an example of
 * USART2 Interrupt Service Routine. This function
 * will be called when a USART2 interrupt occurs
```

```
 * and the driver is configured for interrupt mode
 * operation */

void __ISR ( _UART_2_VECTOR,ipl4 ) _InterruptHandler_USART ( void )
{
    /* usartModule1 is the System Module Object
     * that was returned by the DRV_USART_Initialize
     * function. */

    DRV_USART_TasksTransmit(usartModule1);
    DRV_USART_TasksReceive(usartModule1);
    DRV_USART_TasksError(usartModule1);

}

/* In case of Polled mode, the tasks routines are
 * invoked from the SYS_Tasks() routine. */

void SYS_Tasks(void)
{
    DRV_USART_TasksTransmit(usartModule1);
    DRV_USART_TasksReceive(usartModule1);
    DRV_USART_TasksError(usartModule1);
}

/* The SYS_Tasks routine is invoked from the main
 * application while(1) loop. */

while(1)
{
    SYS_Tasks();
}
```

Using the USART Driver with DMA

This topic provides information on using the USART Driver with DMA.

Description

To use the USART Driver with DMA, the following should be noted:

- Include `drv_usart_dma.c` in the project. Do not include `drv_usart.c`.
- Include `drv_usart_buffer_queue_dma.c` in the project. Do not include `drv_usart_buffer_queue.c`.
- Initialize the driver to use DMA. Refer to Initializing the USART Driver for details.
- Refer to the DMA System Service section for details on initializing and using the DMA system service in Polling or Interrupt mode
- The DRV_USART_INTERRUPT_MODE configuration macro should be set to 'true'
- Do not directly invoke the DRV_USART_TasksTransmit and DRV_USART_TasksReceive functions

Configuring the Library

Macros

	Name	Description
	DRV_USART_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_USART_INDEX	USART Static Index selection.
	DRV_USART_INTERRUPT_MODE	Macro controls interrupt based operation of the driver.
	DRV_USART_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source for the static driver.
	DRV_USART_PERIPHERAL_ID	Configures the USART PLIB Module ID.
	DRV_USART_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_USART_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_USART_BYTE_MODEL_SUPPORT	Specifies if the Byte Model support should be enabled.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE	Defines the Receive interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA	Defines the Receive DMA Channel interrupt source for the static driver.

	DRV_USART_INTERRUPT_SOURCE_TRANSMIT	Defines the Transmit interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA	Defines the Transmit DMA Channel interrupt source for the static driver.
	DRV_USART_QUEUE_DEPTH_COMBINED	Defines the number of entries of all queues in all instances of the driver.
	DRV_USART_READ_WRITE_MODEL_SUPPORT	Specifies if Read/Write Model support should be enabled.
	DRV_USART_RECEIVE_DMA	Defines the USART Driver Receive DMA Channel for the static driver.
	DRV_USART_TRANSMIT_DMA	Defines the USART Driver Transmit DMA Channel in case of static driver.
	DRV_USART_BAUD_RATE_IDXn	Specifies the USART Baud at which the USART driver is initialized.
	DRV_USART_BYTE_MODEL_BLOCKING	Enables or Disables DRV_USART_ByteWrite function blocking behavior.
	DRV_USART_BYTE_MODEL_CALLBACK	Enables or Disables Callback Feature of the Byte Transfer Model.
	DRV_USART_RCV_QUEUE_SIZE_IDXn	Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.
	DRV_USART_XMIT_QUEUE_SIZE_IDXn	Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

Description

The USART Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

Note: Initialization overrides are not supported in this version.

```
/* In this configuration example, the USART driver
 * must manage only on USART peripheral instance.
 * This macro can be greater than one if more
 * USART peripherals are needed. Not defining this
 * macro will cause the driver to be built in
 * static mode */
#define DRV_USART_INSTANCES_NUMBER 1

/* There will be 3 different client that use the
 * one instance of the USART peripheral. Note that
 * this macro configures the total (combined) number of clients
 * across all instance of the USART driver. Not defining
 * this macro will cause the driver to be configured
 * for single client operation */
#define DRV_USART_CLIENTS_NUMBER 3

/* USART Driver should be built for interrupt mode.
 * Set this to false for Polled mode operation */
#define DRV_USART_INTERRUPT_MODE true

/* Combined buffer queue depth is 5. Refer to the
 * description of the Buffer Queue data transfer model
 * and the DRV_USART_QUEUE_DEPTH_COMBINED macro
 * for more details on how this is configured. */
#define DRV_USART_QUEUE_DEPTH_COMBINED 5

/* Set this macro to true is Buffer Queue data
 * transfer model is to be enabled. */
#define DRV_USART_BUFFER_QUEUE_SUPPORT true

/* Set this macro to true if Byte by Byte data
 * transfer model is to be enabled. */
#define DRV_USART_BYTE_MODEL_SUPPORT false

/* Set this macro to true File IO type Read Write
 * data transfer model is to be enabled */
#define DRV_USART_READ_WRITE_MODEL_SUPPORT false
```

DRV_USART_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_usart_config_template.h

C

```
#define DRV_USART_CLIENTS_NUMBER 4
```

Description

USART Client Count Configuration

This macro sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if USART1 will be accessed by two clients and USART2 will accessed by three clients, this number should be 5. It is recommended that this value be set exactly equal to the number of expected clients, as client support consumes RAM memory space. If this macro is not defined and the DRV_USART_INSTANCES_NUMBER macro is not defined, the driver will be built for static - single client operation. If this macro is defined and the DRV_USART_INSTANCES_NUMBER macro is not defined, the driver will be built for static - multi client operation.

Remarks

None.

DRV_USART_INDEX Macro

USART Static Index selection.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INDEX DRV_USART_INDEX_2
```

Description

Index - Used for static drivers

USART Static Index selection for the driver object reference. This macro defines the driver index for static and static multi-client builds. For example, if this macro is set to DRV_USART_INDEX_2, the static driver APIs would be DRV_USART2_Initialize, DRV_USART2_Open, etc. When building static drivers, this macro should be different for each static build of the USART driver that needs to be included in the project.

Remarks

This index is required to make a reference to the driver object

DRV_USART_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_MODE true
```

Description

USART Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values are:

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, the Interrupt Service Routine (ISR) for the interrupt should be defined in the system. The DRV_USART_Tasks routine should be called in the ISR. While using the USART driver with DMA, this flag should always be true.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_ERROR Macro

Defines the error interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_ERROR INT_SOURCE_USART_2_ERROR
```

Description

Error Interrupt Source

This macro defines the Error interrupt source for the static driver. The interrupt source defined by this macro will override the errorInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module error interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_PERIPHERAL_ID Macro

Configures the USART PLIB Module ID.

File

drv_usart_config_template.h

C

```
#define DRV_USART_PERIPHERAL_ID USART_ID_2
```

Description

USART Peripheral Library Module ID

This macro configures the PLIB ID if the driver is built statically. This value will override the usartID member of the DRV_USART_INIT initialization data structure. In that when the driver is built statically, the usartID member of the DRV_USART_INIT data structure will be ignored by the driver initialization routine and this macro will be considered. This should be set to the PLIB ID of USART module (USART_ID_1, USART_ID_2, and so on).

Remarks

None.

DRV_USART_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INSTANCES_NUMBER 2
```

Description

USART driver objects configuration

This macro sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of USART modules that are needed by the application, as hardware Instance support consumes RAM memory space. If this macro is not defined, the driver will be built statically.

Remarks

None

DRV_USART_BUFFER_QUEUE_SUPPORT Macro

Specifies if the Buffer Queue support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BUFFER_QUEUE_SUPPORT true
```

Description

USART Driver Buffer Queue Support

This macro defines whether or not Buffer Queue support should be enabled. Setting this macro to true will enable buffer queue support and all buffer related driver function. The driver should be built along with the drv_usart_buffer_queue.c file, which contains the functional implementation for buffer queues. If buffer queue operation is enabled, the DRV_USART_BYTE_MODEL_SUPPORT function should not be true. If this macro is set to false, the behavior of the USART Driver Buffer Queue API is not defined. While using the USART driver with DMA, the driver supports Buffer Queue Data transfer model regardless of the value of this configuration macro.

Remarks

None.

DRV_USART_BYTE_MODEL_SUPPORT Macro

Specifies if the Byte Model support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_SUPPORT false
```

Description

USART Driver Byte Model Support

This macro defines whether or Byte Model support should be enabled. Setting this macro to true will enable byte model support and all byte operation related driver functions. The driver should be built along with the drv_usart_byte_model.c file, which contains the functional implementation for byte model operation. If byte model operation is enabled, the driver will not support buffer queue and read write models. The behavior of the byte mode API when this macro is set to false is not defined.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_RECEIVE Macro

Defines the Receive interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_RECEIVE INT_SOURCE_USART_2_RECEIVE
```

Description

Receive Interrupt Source

This macro defines the Receive interrupt source for the static driver. The interrupt source defined by this macro will override the rxInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module receive interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA Macro

Defines the Receive DMA Channel interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA
```

Description

Receive DMA Channel Interrupt Source

This macro defines the Receive DMA Channel interrupt source for the static driver. The interrupt source defined by this macro will override the dmaInterruptReceive member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_TRANSMIT Macro

Defines the Transmit interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_TRANSMIT INT_SOURCE_USART_2_TRANSMIT
```

Description

Transmit Interrupt Source

This macro defines the TX interrupt source for the static driver. The interrupt source defined by this macro will override the txInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module transmit interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA Macro

Defines the Transmit DMA Channel interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA
```

Description

Transmit DMA Channel Interrupt Source

This macro defines the TX DMA Channel interrupt source for the static driver. The interrupt source defined by this macro will override the dmaInterruptTransmit member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_QUEUE_DEPTH_COMBINED Macro

Defines the number of entries of all queues in all instances of the driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_QUEUE_DEPTH_COMBINED 16
```

Description

USART Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_USART_BufferAddWrite function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_USART_BufferAddRead function.

A buffer queue will contain buffer queue entries, with each related to a BufferAdd request. This configuration macro defines the total number of buffer entries that will be available for use between all USART driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. The greater the number of buffer entries, the greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

For example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Therefore, the total number of buffer entries should be 2.

As another example, consider the case of a dynamic driver (i.e., two instances) where instance 1 will queue up to three write requests and up to two read requests, and instance 2 will queue up to two write requests and up to six read requests, the value of this macro should be: 13 (2 + 3 + 2 + 6).

Remarks

None.

DRV_USART_READ_WRITE_MODEL_SUPPORT Macro

Specifies if Read/Write Model support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_READ_WRITE_MODEL_SUPPORT true
```

Description

USART Driver Read Write Model Support

This macro defines whether or not Read Write Model support should be enabled. Setting this macro to true will enable read write model support and all read/write related driver functions. The driver should be built along with the drv_usart_read_write.c file, which contains the functional implementation for byte model operation. If read/write model operation is enabled, the DRV_USART_BYTE_MODEL_SUPPORT macro should not be true. The behavior of the Read Write Model API when this macro is set to false is not defined.

Remarks

None.

DRV_USART_RECEIVE_DMA Macro

Defines the USART Driver Receive DMA Channel for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_RECEIVE_DMA
```

Description

USART Driver Receive DMA Channel

This macro defines the USART Receive DMA Channel for the static driver. The DMA channel defined by this macro will override the dmaReceive member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the

DMA PLIB for the microcontroller.

Remarks

None.

DRV_USART_TRANSMIT_DMA Macro

Defines the USART Driver Transmit DMA Channel in case of static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_TRANSMIT_DMA
```

Description

USART Driver Transmit DMA Channel

This macro defines the USART Transmit DMA Channel for the static driver. The DMA channel defined by this macro will override the dmaTransmit member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_USART_BAUD_RATE_IDXn Macro

Specifies the USART Baud at which the USART driver is initialized.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BAUD_RATE_IDXn
```

Description

USART Driver Baud Selection.

This configuration constant specifies the baud rate at which the USART Driver is initialized. This is the baud rate at which the USART module will operate when the driver initialization has completed. The driver client can call the DRV_USART_BaudSet function after opening the driver to change the USART baud rate after initialization has completed.

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Baud Selection field.

DRV_USART_BYTE_MODEL_BLOCKING Macro

Enables or Disables DRV_USART_ByteWrite function blocking behavior.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_BLOCKING
```

Description

USART Driver Byte Write Blocking Behavior

This USART Driver MHC option controls the blocking behavior of the DRV_USART_ByteWrite function and is only applicable when the USART Driver Byte Transfer model is selected. Selecting this option will cause the DRV_USART_ByteWrite function to block until the byte has been written to the USART Transmit FIFO. Blocking behavior is enabled by default (to enable backward compatibility with previous versions of the driver). This option can be used for simple applications where interoperability with other MPLAB Harmony modules is not a design concern.

If the application uses several other MPLAB Harmony modules (Middleware, File System, etc.), it is recommended to disable this option and use the non-blocking DRV_USART_ByteWrite function. This requires the application to call the DRV_USART_TransmitBufferIsFull function to check if the byte can be written to the USART, as shown in the following code example.

```
if(!DRV_USART_TransmitBufferIsFull(usartHandle1))
{
    byte = '1';
    DRV_USART_WriteByte(usartHandle1,byte);
}
```

Using the non-blocking implementation results in improved application interoperability with other MPLAB Harmony modules.

Remarks

The DRV_USART_BYTE_MODEL_BLOCKING constant is specified for documentation purposes only. It does not affect the configuration of the driver.

DRV_USART_BYTE_MODEL_CALLBACK Macro

Enables or Disables Callback Feature of the Byte Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_CALLBACK
```

Description

USART Driver Byte Model Callback Feature.

This USART Driver MHC option controls the Callback feature of the Byte Transfer model. Selecting this option allows an application to register Byte Transfer Event Callback functions with the driver. These callback functions are invoked on the occurrence of Byte Transfer events. Callback functions can be registered to Byte Transmit, Byte Receive, and USART Error events, as shown in the following code example.

```
// This code shows how a callback function is
// registered for the Byte Receive event.
DRV_USART_ByteReceiveCallbackSet(DRV_USART_INDEX_0, APP_USARTReceiveEventHandler);

// Event Processing Technique. Event is received when
// a byte is received.

void APP_USARTReceiveEventHandler(const SYS_MODULE_INDEX index)
{
    // Byte has been Received. Handle the event.
    // Read byte using DRV_USART_ReadByte.
}
```

When operating in Interrupt mode, the callback functions are invoked in an interrupt context. If this option is not selected, the application must use the DRV_USART_TransmitBufferIsFull, DRV_USART_ReceiverBufferIsEmpty, and DRV_USART_ErrorGet functions to check the status of Byte transmit or receive.

Remarks

The DRV_USART_BYTE_MODEL_CALLBACK constant is specified for documentation purposes only. It does not affect the configuration of the driver.

DRV_USART_RCV_QUEUE_SIZE_IDXn Macro

Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_RCV_QUEUE_SIZE_IDXn
```

Description

USART Driver Receive Queue Size Selection.

This constant sets the USART Driver Receive queue size when using the Buffer Queue Data Transfer Model. It affects the queuing capacity of the DRV_USART_BufferAddRead function for the selected driver instance. For example, if this option is set to 5 for USART Driver 0, USART Driver 0 can then queue up to a maximum of five driver client receive buffer requests from any driver clients.

Therefore, if USART Driver 0 has two clients and if client 1 has queued up three buffers for receive, client 2 can only queue up to two buffers. If the client attempts to queue up more buffers, DRV_USART_BufferAddRead will not accept the request and will generate an invalid buffer handle (DRV_USART_BUFFER_HANDLE_INVALID).

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Receive Queue Size field.

DRV_USART_XMIT_QUEUE_SIZE_IDXn Macro

Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_XMIT_QUEUE_SIZE_IDXn
```

Description

USART Driver Transmit Queue Size Selection.

This constant sets the USART Driver Transmit queue size when using the Buffer Queue Data Transfer Model. It affects the queuing capacity of the DRV_USART_BufferAddWrite function, for the selected driver instance. For example, if this option is set to 5 for USART Driver 0, USART Driver 0 can then queue up to a maximum of five driver client transmit buffer requests from any driver clients.

Therefore if USART Driver 0 has two clients and if client 1 has queued up three buffers for transmit, client 2 can only queue up to two buffers. If the client attempts to queue up more buffers, DRV_USART_BufferAddWrite will not accept the request and will generate an invalid buffer handle (DRV_USART_BUFFER_HANDLE_INVALID).

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Transmit Queue Size field.

Building the Library

This section lists the files that are available in the USART Driver Library.

Description

This section list the files that are available in the \src folder of the USART Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usart`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_usart.h`	This file should be included by any .c file which accesses the USART Driver API. This one file contains the prototypes for all driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_usart.c`	This file should always be included in the project when using the USART Driver.
`/src/dynamic/drv_usart_dma.c`	This file should always be included in the project when using the USART driver with DMA.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_usart_byte_model.c`	This file should be included in the project if the USART Driver Byte Model API is required.
`/src/dynamic/drv_usart_buffer_queue.c`	This file should be included in the project if the USART Driver Buffer Queue Model API (without DMA) is required.

/src/dynamic/drv_usart_read_write.c	This file should be included in the project if the USART Driver Read Write Model API is required.
/src/dynamic/drv_usart_buffer_queue_dma.c	This file should be included in the project if the USART Driver Buffer Queue Model API with DMA is required.

Module Dependencies

The USART Driver Library depends on the following modules:

- Interrupt System Service Library
- DMA System Service Library (if USART Driver is configured to use DMA)

Library Interface

a) System Functions

	Name	Description
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_Status	Gets the current status of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_TasksReceive	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksTransmit	Maintains the driver's transmit state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Static/Dynamic

b) Core Client Functions

	Name	Description
	DRV_USART_Open	Opens the specified USART driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_USART_Close	Closes an opened-instance of the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_ClientStatus	Gets the current client-specific status the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Static/Dynamic

c) Communication Management Client Functions

	Name	Description
	DRV_USART_BaudSet	This function changes the USART module baud to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_LineControlSet	This function changes the USART module line control to the specified value. **Implementation:** Static/Dynamic

d) Buffer Queue Read/Write Client Functions

	Name	Description
	DRV_USART_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Static/Dynamic
	DRV_USART_BufferProcessedSizeGet	This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.

	DRV_USART_AddressedBufferAddWrite	Schedule a non-blocking addressed driver write operation. **Implementation:** Dynamic
	DRV_USART_BufferCompletedBytesGet	Returns the number of bytes that have been processed for the specified buffer. **Implementation:** Static/Dynamic
	DRV_USART_BufferRemove	Removes a requested buffer from the queue. **Implementation:** Static/Dynamic

e) File I/O Type Read/Write Functions

	Name	Description
	DRV_USART_Read	Reads data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_Write	Writes data to the USART. **Implementation:** Static/Dynamic

f) Byte Transfer Functions

	Name	Description
	DRV_USART_ReadByte	Reads a byte of data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_WriteByte	Writes a byte of data to the USART. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferSizeGet	Returns the size of the transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferSizeGet	Returns the size of the receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_TransferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferIsFull	Provides the status of the driver's transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferIsEmpty	Provides the status of the driver's receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_ByteErrorCallbackSet	Registers callback to handle for byte error events.
	DRV_USART_ByteReceiveCallbackSet	Registers receive callback function for byte receive event.
	DRV_USART_ByteTransmitCallbackSet	Registers a callback function for byte transmit event.

Description

This section describes the functions of the USART Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_USART_Initialize Function

Initializes the USART instance for the specified driver index.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
SYS_MODULE_OBJ DRV_USART_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the USART driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the USART module ID. For example, driver instance 0 can be assigned to

USART2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_USART_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other USART routine is called.

This routine should only be called once during system initialization unless DRV_USART_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
// The following code snippet shows an example USART driver initialization.
// The driver is initialized for normal mode and a baud of 300. The
// receive queue size is set to 2 and transmit queue size is set to 3.

DRV_USART_INIT              usartInit;
SYS_MODULE_OBJ              objectHandle;

usartInit.baud  = 300;
usartInit.mode  = DRV_USART_OPERATION_MODE_NORMAL;
usartInit.flags = DRV_USART_INIT_FLAG_NONE;
usartInit.usartID   = USART_ID_2;
usartInit.brgClock  = 80000000;
usartInit.handshake = DRV_USART_HANDSHAKE_NONE;
usartInit.lineControl       = DRV_USART_LINE_CONTROL_8NONE1;
usartInit.interruptError    = INT_SOURCE_USART_2_ERROR;
usartInit.interruptReceive  = INT_SOURCE_USART_2_RECEIVE;
usartInit.queueSizeReceive  = 2;
usartInit.queueSizeTransmit = 3;
usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;
usartInit.moduleInit.value  = SYS_MODULE_POWER_RUN_FULL;

objectHandle = DRV_USART_Initialize(DRV_USART_INDEX_1, (SYS_MODULE_INIT*)&usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_USART_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

)

DRV_USART_Deinitialize Function

Deinitializes the specified instance of the USART driver module.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the USART driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_USART_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_USART_Initialize
SYS_STATUS          status;

DRV_USART_Deinitialize(object);

status = DRV_USART_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Function

void DRV_USART_Deinitialize(SYS_MODULE_OBJ object)

DRV_USART_Status Function

Gets the current status of the USART driver module.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
SYS_STATUS DRV_USART_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

Description

This routine provides the current status of the USART driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_USART_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize
SYS_STATUS          usartStatus;

usartStatus = DRV_USART _Status(object);
```

```
if (SYS_STATUS_READY == usartStatus)
{
    // This means the driver can be opened using the
    // DRV_USART_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Function

SYS_STATUS DRV_USART_Status(SYS_MODULE_OBJ object)

DRV_USART_TasksReceive Function

Maintains the driver's receive state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksReceive(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal receive state machine and implement its receive ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the receive interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksReceive (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksReceive (SYS_MODULE_OBJ object);

DRV_USART_TasksTransmit Function

Maintains the driver's transmit state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksTransmit(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmit state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksTransmit (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksTransmit (SYS_MODULE_OBJ object);

DRV_USART_TasksError Function

Maintains the driver's error state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksError(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal error state machine and implement its error ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the error interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksError (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksError (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_USART_Open Function

Opens the specified USART driver instance and returns a handle to it.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_HANDLE DRV_USART_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_USART_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver is not ready to be opened, typically when the initialize routine has not completed execution.

Description

This routine opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_USART_Read and DRV_USART_Write functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed. If the driver is configured for polling and bare-metal operation, it will not support DRV_IO_INTENT_BLOCKING. The driver will operation will always be non-blocking.

If ioIntent is DRV_IO_INTENT_READ, the client will only be able to read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent is DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_USART_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_USART_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_USART_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
)

DRV_USART_Close Function

Closes an opened-instance of the USART driver.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the USART driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines (with one possible exception described in the "Remarks" section). A new handle must be obtained by calling DRV_USART_Open before the caller may use the driver again

Remarks

Usually there is no need for the client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called. However, if it requires additional time to do so in a non-blocking environment, it will still return from the Close operation but the handle is now a zombie handle. The client can only call the DRV_USART_ClientStatus on a zombie handle to track the completion of the Close operation. The DRV_USART_ClientStatus routine will return DRV_CLIENT_STATUS_CLOSED when the close operation has completed.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USART_Open

DRV_USART_Close(handle);
```

```
// After this point, the handle cannot be used with any other function
// except the DRV_USART_ClientStatus function, which can be used to query
// the success status of the DRV_USART_Close function.

while(DRV_USART_CLIENT_STATUS_CLOSED != DRV_USART_ClientStatus(handle));
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_USART_Close(DRV_Handle handle)

DRV_USART_ClientStatus Function

Gets the current client-specific status the USART driver.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_CLIENT_STATUS DRV_USART_ClientStatus(DRV_HANDLE handle);
```

Returns

A DRV_USART_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the USART driver associated with the given handle. This function can be used to check the status of client after the DRV_USART_Close() function has been called.

Remarks

This function will not block for hardware access and will immediately return the current status. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize function must have been called.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE              handle;  // Returned from DRV_USART_Open
DRV_USART_CLIENT_STATUS   status;

status = DRV_USART_ClientStatus(handle);
if( DRV_USART_CLIENT_STATUS_CLOSED != status )
{
    // The client had not closed.
}
```

Parameters

Parameters	Description
handle	Handle returned from the driver's open function.

Function

DRV_USART_CLIENT_STATUS DRV_USART_ClientStatus(DRV_HANDLE handle)

DRV_USART_ErrorGet Function

This function returns the error(if any) associated with the last client request.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_ERROR DRV_USART_ErrorGet(const DRV_HANDLE client);
```

Returns

A DRV_USART_ERROR type indicating last known error status.

Description

This function returns the error(if any) associated with the last client request. DRV_USART_Read and DRV_USART_Write will update the client error status when these functions return DRV_USART_TRANSFER_ERROR. If the driver send a DRV_USART_BUFFER_EVENT_ERROR to the client, the client can call this function to know the error cause. The error status will be updated on every operation and should be read frequently (ideally immediately after the driver operation has completed) to know the relevant error status.

Remarks

It is the client's responsibility to make sure that the error status is obtained frequently. The driver will update the client error status regardless of whether this has been examined by the client. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once.

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARThandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:
```

```
            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred. We can also find
            // the error cause.

            processedBytes = DRV_USART_BufferCompletedBytesGet(bufferHandle);
            if(DRV_USART_ERROR_RECEIVE_OVERRUN == DRV_USART_ErrorGet(myUSARTHandle))
            {
                // There was an receive over flow error.
                // Do error handling here.
            }

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

DRV_USART_ERROR DRV_USART_ErrorGet(DRV_HANDLE client);

c) Communication Management Client Functions

DRV_USART_BaudSet Function

This function changes the USART module baud to the specified value.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_BAUD_SET_RESULT DRV_USART_BaudSet(const DRV_HANDLE client, uint32_t baud);
```

Returns

None.

Description

This function changes the USART module baud to the specified value. Any queued buffer requests will be processed at the updated baud. The USART driver operates at the baud specified in DRV_USART_Initialize function unless the DRV_USART_BaudSet function is called to change the baud.

Remarks

The implementation of this function, in this release of the driver, changes the baud immediately. This may interrupt on-going data transfer. It is recommended that the driver be opened exclusively if this function is to be called. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

DRV_USART_BaudSet(myUSARTHandle, 9600);
```

Parameters

Parameters	Description
handle	client handle returned by DRV_USART_Open function.
baud	desired baud.

Function

void DRV_USART_BaudSet(DRV_HANDLE client, uint32_t baud);

DRV_USART_LineControlSet Function

This function changes the USART module line control to the specified value.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_LINE_CONTROL_SET_RESULT DRV_USART_LineControlSet(const DRV_HANDLE client, const
DRV_USART_LINE_CONTROL lineControl);
```

Returns

DRV_USART_LINE_CONTROL_SET_SUCCESS if the function was successful. Returns DRV_HANDLE_INVALID if the client handle is not valid.

Description

This function changes the USART module line control parameters to the specified value. Any queued buffer requests will be processed at the updated line control parameters. The USART driver operates at the line control parameters specified in DRV_USART_Initialize function unless the DRV_USART_LineControlSet function is called to change the line control parameters.

Remarks

The implementation of this function, in this release of the driver, changes the line control immediately. This may interrupt on-going data transfer. It is recommended that the driver be opened exclusively if this function is to be called. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

DRV_USART_LineControlSet(myUSARTHandle, DRV_USART_LINE_CONTROL_8NONE1);
```

Parameters

Parameters	Description
handle	client handle returned by DRV_USART_Open function.
lineControl	line control parameters.

Function

void DRV_USART_LineControlSet

(

DRV_HANDLE client,

DRV_USART_LINE_CONTROL lineControl

);

d) Buffer Queue Read/Write Client Functions

DRV_USART_BufferAddRead Function

Schedule a non-blocking driver read operation.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferAddRead(const DRV_HANDLE handle, DRV_USART_BUFFER_HANDLE * const bufferHandle, void *
buffer, const size_t size);
```

Returns

The buffer handle is returned in the bufferHandle argument. This is DRV_USART_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_USART_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by the client. It should not be called in the event handler associated with another USART driver instance. It should not be called directly in an ISR.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_BufferAddRead(myUSARThandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
```

```
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the communication channel as returned by the DRV_USART_Open function.
buffer	Buffer where the received data will be stored.
size	Buffer size in bytes.

Function

void DRV_USART_BufferAddRead

(

const DRV_HANDLE handle,

DRV_USART_BUFFER_HANDLE * bufferHandle,

void * buffer,

const size_t size

)

DRV_USART_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferAddWrite(const DRV_HANDLE handle, DRV_USART_BUFFER_HANDLE * bufferHandle, void *
buffer, const size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_USART_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. On returning, the bufferHandle parameter may be DRV_USART_BUFFER_HANDLE_INVALID for the following reasons:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0

- if the transmit queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or a DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another USART driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_BufferAddWrite(myUSARThandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the communication channel as return by the DRV_USART_Open function.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

buffer	Data to be transmitted.
size	Buffer size in bytes.

Function

void DRV_USART_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_USART_BUFFER_HANDLE * bufferHandle,
void * buffer,
size_t size
);

DRV_USART_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferEventHandlerSet(const DRV_HANDLE handle, const DRV_USART_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

DRV_USART_BufferAddRead(myUSARThandle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
```

```
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_USART_BufferEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_USART_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t context

)

DRV_USART_BufferProcessedSizeGet Function

This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.

File

drv_usart.h

C

```
size_t DRV_USART_BufferProcessedSizeGet(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

None.

Description

None.

Remarks

None.

Preconditions

None.

Example

None.

Function

size_t DRV_USART_BufferProcessedSizeGet

(

DRV_USART_BUFFER_HANDLE bufferHandle

);

DRV_USART_AddressedBufferAddWrite Function

Schedule a non-blocking addressed driver write operation.

Implementation: Dynamic

File

drv_usart.h

C

```
void DRV_USART_AddressedBufferAddWrite(const DRV_HANDLE hClient, DRV_USART_BUFFER_HANDLE * bufferHandle,
uint8_t address, void * source, size_t nWords);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_USART_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking addressed write operation. The function returns with a valid buffer handle in the bufferHandle argument if the addressed write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. On returning, the bufferHandle parameter may be DRV_USART_BUFFER_HANDLE_INVALID for the following reasons:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0
- if the transmit queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or a DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another USART driver instance. It should not otherwise be called directly in an ISR.

The source buffer should be a 16-bit word aligned buffer. The 9th bit of the higher byte 16-bit buffer is used to indicate data/address.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

The operation mode of the driver must be DRV_USART_OPERATION_MODE_ADDRESSED.

Example

```
MY_APP_OBJ myAppObj;
uint16_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;
```

```
uint8_t clientAddress;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

clientAddress = 0x60;
DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_AddressedBufferAddWrite(myUSARThandle, &bufferHandle, clientAddress
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
hClient	Handle of the communication channel as return by the DRV_USART_Open function.
bufferHandle	Pointer to an argument that will contain the return buffer handle.
address	Address of the receiver client
source	Data to be transmitted.
size	Buffer size in 16-bit words.

Function

void DRV_USART_AddressedBufferAddWrite
(
const DRV_HANDLE hClient,
DRV_USART_BUFFER_HANDLE * bufferHandle,
uint8_t address,
void * source,
size_t nWords
);

DRV_USART_BufferCompletedBytesGet Function

Returns the number of bytes that have been processed for the specified buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_BufferCompletedBytesGet(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

Returns the number of bytes that have been processed for this buffer.

Returns DRV_USART_BUFFER_HANDLE_INVALID for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. Or in any other use case. This function can be used for non-DMA buffer transfers only. It cannot be used when the USART driver is configured to use DMA.

Remarks

This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARTHandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
```

```
            case DRV_USART_BUFFER_EVENT_COMPLETE:

                // This means the data was transferred.
                break;

            case DRV_USART_BUFFER_EVENT_ERROR:

                // Error handling here.
                // We can find out how many bytes were processed in this
                // buffer before the error occurred.

                processedBytes = DRV_USART_BufferCompletedBytesGet(bufferHandle);

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
bufferhandle	Handle for the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_USART_BufferCompletedBytesGet
(
DRV_USART_BUFFER_HANDLE bufferHandle
);

DRV_USART_BufferRemove Function

Removes a requested buffer from the queue.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_BUFFER_RESULT DRV_USART_BufferRemove(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

DRV_USART_BUFFER_RESULT_HANDLE_INVALID - Buffer handle is invalid.

DRV_USART_BUFFER_RESULT_HANDLE_EXPIRED - Buffer handle is expired.

DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY - Buffer is removed from the queue successfully.

DRV_USART_BUFFER_RESULT_REMOVAL_FAILED - Failed to remove buffer from the queue because of mutex timeout in RTOS environment.

Description

This function removes a specified buffer from the queue. The client can use this function to delete

1. An unwated stalled buffer.
2. Queued buffers on timeout.

or in any other use case.

Remarks

This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARThandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}

// Timeout function, where remove queued buffer if it still exists.
void APP_TimeOut(void)
{
    DRV_USART_BUFFER_RESULT bufferResult;
    bufferResult = DRV_USART_BufferRemove(bufferHandle);

    if(DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY == bufferResult)
    {
        //Buffer removed succesfully from the queue
    }
    else
    {
        //Either buffer is invalid or expired.
        //Or not able to acquire mutex in RTOS mode.
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer to delete.

Function

DRV_USART_BUFFER_RESULT DRV_USART_BufferRemove(DRV_USART_BUFFER_HANDLE bufferHandle)

e) File I/O Type Read/Write Functions

DRV_USART_Read Function

Reads data from the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_Read(const DRV_HANDLE handle, void * buffer, const size_t numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer. Returns DRV_USART_READ_ERROR in case of an error.

Description

This routine reads data from the USART. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been received or if an error occurred. If there are buffers queued for receiving data, these buffers will be serviced first. The function will not return until the requested number of bytes have been read.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually read. The function will not wait until numBytes of bytes have been read. If there are buffer queued for reading data, then the function will not block and will return immediately with 0 bytes read.

Remarks

This function is thread safe in a RTOS application. If the driver is configured for polled operation, this it will not support blocking operation in a bare metal (non-RTOS) application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_USART_Read(myUSARTHandle, &myBuffer[total], MY_BUFFER_SIZE - total);
    if(count == DRV_USART_READ_ERROR)
    {
       // There was an error. The DRV_USART_ErrorGet() function
       // can be called to find the exact error.
    }
    total += count;

    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer into which the data read from the USART instance will be placed.
numbytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Function

size_t DRV_USART_Read
(
const DRV_HANDLE handle,
void * buffer,
const size_t numbytes
)

DRV_USART_Write Function

Writes data to the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_Write(const DRV_HANDLE handle, void * buffer, const size_t numbytes);
```

Returns

Number of bytes actually written to the driver. Return DRV_USART_WRITE_ERROR in case of an error.

Description

This routine writes data to the USART. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been transmitted or if an error occurred. If there are buffers queued for writing, the function will wait until all the preceding buffers are completed. Ongoing buffer transmit operations will not be affected.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually accepted for transmission. The function will not wait until numBytes of bytes have been transmitted. If there a buffers queued for transmit, the function will not wait and will return immediately with 0 bytes.

Remarks

This function is thread safe in a RTOS application. This function is thread safe in a RTOS application. If the driver is configured for polled operation, this it will not support blocking operation in a bare metal (non-RTOS) application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_USART_Write(myUSARTHandle, &myBuffer[total],
                                            MY_BUFFER_SIZE - total);
    total += count;
```

```
    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer containing the data to written.
numbytes	size of the buffer

Function

size_t DRV_USART_Write
(
const DRV_HANDLE handle,
void * buffer,
const size_t numbytes
)

f) Byte Transfer Functions

DRV_USART_ReadByte Function

Reads a byte of data from the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
uint8_t DRV_USART_ReadByte(const DRV_HANDLE handle);
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the USART.

Remarks

This function is thread safe when called in a RTOS application. Note that DRV_USART_WriteByte and DRV_USART_ReadByte function cannot co-exist with DRV_USART_BufferAddRead, DRV_USART_BufferAddWrite, DRV_USART_Read and DRV_USART_Write functions in a application. Calling the DRV_USART_ReadByte and DRV_USART_WriteByte functions will disrupt the processing of any queued buffers.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT & DRV_USART_TransferStatus(myUSARTHandle) )
    {
        myBuffer[numBytes++] = DRV_USART_ReadByte(myUSARTHandle);
    }

    // Do something else...
```

```
} while( numBytes < MY_BUFFER_SIZE);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint8_t DRV_USART_ReadByte(const DRV_HANDLE handle)

DRV_USART_WriteByte Function

Writes a byte of data to the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_WriteByte(const DRV_HANDLE handle, const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the USART.

Remarks

This function is thread safe when called in a RTOS application. Note that DRV_USART_WriteByte and DRV_USART_ReadByte function cannot co-exist with DRV_USART_BufferAddRead, DRV_USART_BufferAddWrite, DRV_USART_Read and DRV_USART_Write functions in a application. Calling the DRV_USART_ReadByte and DRV_USART_WriteByte function will disrupt the processing of any queued buffers.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Preinitialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE );
{
    if( !(DRV_USART_TRANSFER_STATUS_TRANSMIT_FULL & DRV_USART_TransferStatus(myUSARTHandle)) )
    {
        DRV_USART_WriteByte(myUSARTHandle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
byte	Data byte to write to the USART

Function

void DRV_USART_WriteByte(const DRV_HANDLE handle, const uint8_t byte)

DRV_USART_TransmitBufferSizeGet Function

Returns the size of the transmit buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
unsigned int DRV_USART_TransmitBufferSizeGet(const DRV_HANDLE handle);
```

Returns

Size of the driver's transmit buffer, in bytes.

Description

This routine returns the size of the transmit buffer and can be used by the application to determine the number of bytes to write with the DRV_USART_WriteByte function.

Remarks

Does not account for client queued buffers. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
const uint8_t   writeBuffer[5];
unsigned int    size, numBytes = 0;
unsigned int    writeBufferLen = sizeof(writeBuffer);

size     = DRV_USART_TransmitBufferSizeGet (myUSARTHandle);

// Do something based on the transmitter buffer size
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

unsigned int DRV_USART_TransmitBufferSizeGet (const DRV_HANDLE handle)

DRV_USART_ReceiverBufferSizeGet Function

Returns the size of the receive buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
unsigned int DRV_USART_ReceiverBufferSizeGet(const DRV_HANDLE handle);
```

Returns

Size of the driver's receive buffer, in bytes.

Description

This routine returns the size of the receive buffer.

Remarks

Does not account for client queued buffers. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
const uint8_t   readBuffer[5];
unsigned int    size, numBytes = 0;
unsigned int    readbufferLen = sizeof(readBuffer);

size     = DRV_USART_ReceiverBufferSizeGet(myUSARTHandle);

// Do something based on the receiver buffer size
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

unsigned int DRV_USART_ReceiverBufferSizeGet(const DRV_HANDLE handle)

DRV_USART_TransferStatus Function

Returns the transmitter and receiver transfer status.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_TRANSFER_STATUS DRV_USART_TransferStatus(const DRV_HANDLE handle);
```

Returns

A DRV_USART_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_USART_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open

if (DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT & DRV_USART_TransferStatus(myUSARTHandle))
{
    // Data has been received that can be read
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_USART_TRANSFER_STATUS DRV_USART_TransferStatus(const DRV_HANDLE handle)

DRV_USART_TransmitBufferIsFull Function

Provides the status of the driver's transmit buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
bool DRV_USART_TransmitBufferIsFull(const DRV_HANDLE handle);
```

Returns

true - if the transmit buffer is full

false - if the transmit buffer is not full

Description

This routine identifies if the driver's transmit buffer is full or not. This function can be used in conjunction with the DRV_USART_Write and DRV_USART_WriteByte functions.

Remarks

Does not account for client queued buffers. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
unsigned int    numBytes;
int             bytesToWrite;
const uint8_t   writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
int             writebufferLen = strlen((char *)writeBuffer);

numBytes = 0;
while( numBytes < writebufferLen )
{
    if (DRV_USART_TransmitBufferisFull())
    {
        // Do something else until there is some room in the driver's Transmit buffer.
    }
    else
    {
        DRV_USART_WriteByte(myUSARTHandle, writeBuffer[numBytes++]);
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_USART_TransmitBufferIsFull(const DRV_HANDLE handle)

DRV_USART_ReceiverBufferIsEmpty Function

Provides the status of the driver's receive buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
bool DRV_USART_ReceiverBufferIsEmpty(const DRV_HANDLE handle);
```

Returns

true - if the driver's receive buffer is empty

false - if the driver's receive buffer is not empty

Description

This routine indicates if the driver's receiver buffer is empty. This function can be used in conjunction with the DRV_USART_Read and DRV_USART_ReadByte functions.

Remarks

Does not account for client queued buffers. This function is safe thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE               myUSARTHandle;    // Returned from DRV_USART_Open
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE );
{
    if ( !DRV_USART_ReceiverBufferIsEmpty(myUSARTHandle) )
    {
        if( numBytes < MY_BUFFER_SIZE )
        {
            myBuffer[numBytes++] = DRV_USART_ReadByte (myUSARTHandle);
        }
        else
        {
            break;
        }
    }

    // Do something else while more data is received.
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_USART_ReceiverBufferIsEmpty(const DRV_HANDLE handle)

DRV_USART_ByteErrorCallbackSet Function

Registers callback to handle for byte error events.

File

drv_usart.h

C

```
void DRV_USART_ByteErrorCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a callback function to be registered with the driver to handle the error events occurring in the transmit/receive path during byte transfers.

The callback function should be registered as part of the initialization. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteErrorCallbackSet (DRV_USART_INDEX_0, APP_USARTErrorEventHandler);

// Event Processing Technique.
void APP_USARTErrorEventHandler(const SYS_MODULE_INDEX index)
{
    // Error has occurred. Handle the event.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteErrorCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

DRV_USART_ByteReceiveCallbackSet Function

Registers receive callback function for byte receive event.

File

drv_usart.h

C

```
void DRV_USART_ByteReceiveCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a receive callback function to be registered with the driver. The callback function is invoked when a byte has been received. The received byte can then be read using DRV_USART_ReadByte() function.

The callback function should be registered with the driver as part of the initialization. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteReceiveCallbackSet(DRV_USART_INDEX_0, APP_USARTReceiveEventHandler);

// Event Processing Technique. Event is received when
// a byte is received.

void APP_USARTReceiveEventHandler(const SYS_MODULE_INDEX index)
{
    // Byte has been Received. Handle the event.
    // Read byte using DRV_USART_ReadByte ()
    // DRV_USART_ReceiverBufferIsEmpty() function can be used to
    // check if the receiver buffer is empty.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteReceiveCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

DRV_USART_ByteTransmitCallbackSet Function

Registers a callback function for byte transmit event.

File

drv_usart.h

C

```
void DRV_USART_ByteTransmitCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a transmit callback function to be registered with the driver. The callback function is invoked when a byte has been transmitted using DRV_USART_WriteByte () function.

The callback function should be registered with the driver prior to any writes to the driver. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteTransmitCallbackSet (DRV_USART_INDEX_0, APP_USARTTransmitEventHandler);

DRV_USART_WriteByte (myUSARThandle, myBuffer[0]);

// Event Processing Technique. Event is received when
// the byte is transmitted.

void APP_USARTTransmitEventHandler (const SYS_MODULE_INDEX index)
{
    // Byte has been transmitted. Handle the event.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteTransmitCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

Files

Files

Name	Description
drv_usart.h	USART Driver Interface Header File
drv_usart_config_template.h	USART Driver Configuration Template.

Description

This section lists the source and header files used by the USART Driver Library.

drv_usart.h

USART Driver Interface Header File

Functions

	Name	Description
	DRV_USART_AddressedBufferAddWrite	Schedule a non-blocking addressed driver write operation. **Implementation:** Dynamic

	DRV_USART_BaudSet	This function changes the USART module baud to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferCompletedBytesGet	Returns the number of bytes that have been processed for the specified buffer. **Implementation:** Static/Dynamic
	DRV_USART_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Static/Dynamic
	DRV_USART_BufferProcessedSizeGet	This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.
	DRV_USART_BufferRemove	Removes a requested buffer from the queue. **Implementation:** Static/Dynamic
	DRV_USART_ByteErrorCallbackSet	Registers callback to handle for byte error events.
	DRV_USART_ByteReceiveCallbackSet	Registers receive callback function for byte receive event.
	DRV_USART_ByteTransmitCallbackSet	Registers a callback function for byte transmit event.
	DRV_USART_ClientStatus	Gets the current client-specific status the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_Close	Closes an opened-instance of the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Static/Dynamic
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_USART_LineControlSet	This function changes the USART module line control to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_Open	Opens the specified USART driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_USART_Read	Reads data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_ReadByte	Reads a byte of data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferIsEmpty	Provides the status of the driver's receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferSizeGet	Returns the size of the receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_Status	Gets the current status of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksReceive	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksTransmit	Maintains the driver's transmit state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TransferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferIsFull	Provides the status of the driver's transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferSizeGet	Returns the size of the transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_Write	Writes data to the USART. **Implementation:** Static/Dynamic
	DRV_USART_WriteByte	Writes a byte of data to the USART. **Implementation:** Static/Dynamic

Description

USART Driver Interface Header File

The USART device driver provides a simple interface to manage the USART or UART modules on Microchip microcontrollers. This file provides the interface definition for the USART driver.

File Name

drv_usart.h

Company

Microchip Technology Inc.

drv_usart_config_template.h

USART Driver Configuration Template.

Macros

	Name	Description
	DRV_USART_BAUD_RATE_IDXn	Specifies the USART Baud at which the USART driver is initialized.
	DRV_USART_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_USART_BYTE_MODEL_BLOCKING	Enables or Disables DRV_USART_ByteWrite function blocking behavior.
	DRV_USART_BYTE_MODEL_CALLBACK	Enables or Disables Callback Feature of the Byte Transfer Model.
	DRV_USART_BYTE_MODEL_SUPPORT	Specifies if the Byte Model support should be enabled.
	DRV_USART_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_USART_INDEX	USART Static Index selection.
	DRV_USART_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_USART_INTERRUPT_MODE	Macro controls interrupt based operation of the driver.
	DRV_USART_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE	Defines the Receive interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA	Defines the Receive DMA Channel interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT	Defines the Transmit interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA	Defines the Transmit DMA Channel interrupt source for the static driver.
	DRV_USART_PERIPHERAL_ID	Configures the USART PLIB Module ID.
	DRV_USART_QUEUE_DEPTH_COMBINED	Defines the number of entries of all queues in all instances of the driver.
	DRV_USART_RCV_QUEUE_SIZE_IDXn	Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.
	DRV_USART_READ_WRITE_MODEL_SUPPORT	Specifies if Read/Write Model support should be enabled.
	DRV_USART_RECEIVE_DMA	Defines the USART Driver Receive DMA Channel for the static driver.
	DRV_USART_TRANSMIT_DMA	Defines the USART Driver Transmit DMA Channel in case of static driver.
	DRV_USART_XMIT_QUEUE_SIZE_IDXn	Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

Description

USART Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_usart_config_template.h

Company

Microchip Technology Inc.

Wi-Fi Driver Libraries

This section describes the Wi-Fi Driver Libraries available in MPLAB Harmony.

Description

MRF24WN0MA Wi-Fi PICtail/PICtail Plus Daughter Board: Part number - AC164153
http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=AC164153

The following table lists the library files available for the Wi-Fi Drivers.

Wi-Fi Library File Matrix

Wi-Fi Device	PIC32MX795F512L	PIC32MZ2048ECH144	PIC32MZ2048EFM144
MRF24WN	wdrvext_mx.a	wdrvext_mz_ec.a	wdrvext_mz_ef.a

MRF24WN Wi-Fi Driver Library

This topic describes the MRF24WN Wi-Fi Driver Library.

Description

The following table lists the library files available for the MRF24WN Wi-Fi Driver.

	Target MCU Device		
Wi-Fi Library File Matrix	PIC32MX795F512L	PIC32MZ2048ECH144	PIC32MZ2048EFM144
Wi-Fi Device MRF24WN	wdrvext_mx.a	wdrvext_mz_ec.a	wdrvext_mz_ef.a

Introduction

This library provides a low-level abstraction of the MRF24WN Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The MRF24WN Wi-Fi Driver Library, in conjunction with the MRF24WN module, allows an application to:

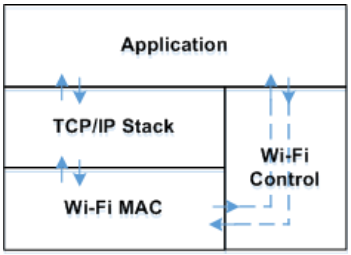
- Join an existing 802.11 Wi-Fi Infrastructure network
- Create a 802.11 Wi-Fi Ad Hoc or Soft AP network

The following application services are provided by the Wi-Fi library:

- Configuring Wi-Fi connection (SSID, security mode, channel list, etc.)
- Join an existing Wi-Fi Infrastructure network
- Create a Wi-Fi Ad Hoc or Soft AP network
- Scan for Wi-Fi Access Point (AP) or Soft AP
- Getting Wi-Fi network status
- Wi-Fi power control
- Wi-Fi console commands

The MAC layer services are not directly accessible to the application; this portion of the code resides under the TCP/IP Stack MAC module software layers and is used by stack services to transmit and receive data over a Wi-Fi network. The following diagram shows the interaction of the primary software blocks in a Wi-Fi application.

Wi-Fi Software Block Diagram



The following table provides information that includes network mode and security mode support by MRF24WN Wi-Fi Driver.

MRF24WN Network Connection Matrix	Network Mode	
	Infrastructure	Soft AP
Security Mode		
Open	YES	YES
WEP40	YES	YES
WEP104	YES	YES
WPA-PSK	YES	YES
WPA2-PSK	YES	YES
WPS Push Button	YES	NA
WPS PIN	YES	NA

Using the Library

This topic describes the basic architecture of the MRF24WN Wi-Fi Driver Library and provides information and examples on its use.

Description

Interface Header Files: `wdrv_mrf24wn_common.h` and `wdrv_mrf24wn_api.h`

The interface to the MRF24WN Wi-Fi Driver Library is defined in the `wdrv_mrf24wn_common.h` and wdrv_mrf24wn_api.h header files.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MRF24WN Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The MRF24WN Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing
- Wi-Fi status
- Wi-FI console commands

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The Library Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the Wi-Fi module.

Library Interface Section	Description
Wi-Fi Initialization Functions	This section provides functions that initialize the Wi-Fi library and allow its API to be used.
Wi-Fi Status Functions	This section provides functions that retrieve the Wi-Fi connection status.
Wi-Fi External Functions	This section provides public functions accessible to TCP/IP applications.
Other Functions	This section provides additional miscellaneous functions for configuring the Wi-Fi connection.

How the Library Works

This section describes how the MRF24WN Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration).

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the MRF24WN Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

System Initialization

This section describes initialization and reinitialization features.

Description

Wi-Fi initialization configures the MRF24WN module and then directs it to join (or create) a Wi-Fi network. The MRF24WN module defaults to open security and scans all channels in the domain. Therefore, to initialize and connect with the minimum function call overhead in an open security network, the following functions can be used:

```
WDRV_EXT_CmdSSIDSet("MySsidName",strlen("MySsidName");
WDRV_EXT_CmdConnect();        // start the connection process
```

Alternatively, the following functions could be used to achieve the same effect:

```
WDRV_EXT_CmdNetModeBSSSet();
WDRV_EXT_CmdSecNoneSet();
WDRV_EXT_CmdSSIDSet("MySsidName",strlen("MySsidName");
WDRV_EXT_CmdConnect();
```

Client Functionality

This section describes core operation.

Description

From the client perspective, once Wi-Fi initialization is complete and the connection process has started, the client responds to Wi-Fi events. The client is notified of events by the callback function WDRV_ProcessEvent. The parameters into that function are `event` and `eventInfo`, where `event` is the event code and `eventInfo` is additional information about the event.

Wi-Fi Connection Events

```
/*No Wi-Fi connection exists*/
WDRV_CSTATE_NOT_CONNECTED = 1,

/*Wi-Fi connection in progress*/
WDRV_CSTATE_CONNECTION_IN_PROGRESS = 2,

/*Wi-Fi connected in infrastructure mode*/
WDRV_CSTATE_CONNECTED_INFRASTRUCTURE = 3,

/*Wi-Fi connected in adHoc mode*/
WDRV_CSTATE_CONNECTED_ADHOC = 4,

/*Wi-Fi in process of reconnecting*/
WDRV_CSTATE_RECONNECTION_IN_PROGRESS = 5,

/*Wi-Fi connection temporarily lost*/
WDRV_CSTATE_CONNECTION_TEMPORARY_LOST = 6,

/*Wi-Fi connection permanently lost*/
WDRV_CSTATE_CONNECTION_PERMANENTLY_LOST = 7
```

Scan Events

```
WDRV_SOFTAP_EVENT_CONNECTED = 0,
WDRV_SOFTAP_EVENT_DISCONNECTED = 1
```

Key Events

```
WDRV_SOFTAP_EVENT_LINK_LOST = 0,
WDRV_SOFTAP_EVENT_RECEIVED_DEAUTH = 1
```

Disconnect Events

```
WDRV_DISCONNECT_REASON_NO_NETWORK_AVAIL = 0x01,
WDRV_DISCONNECT_REASON_LOST_LINK = 0x02,
WDRV_DISCONNECT_REASON_DISCONNECT_CMD = 0x03,
```

```
WDRV_DISCONNECT_REASON_BSS_DISCONNECTED = 0x04,
WDRV_DISCONNECT_REASON_AUTH_FAILED = 0x05,
WDRV_DISCONNECT_REASON_ASSOC_FAILED = 0x06,
WDRV_DISCONNECT_REASON_NO_RESOURCES_AVAIL = 0x07,
WDRV_DISCONNECT_REASON_CONNECTION_DENIED = 0x08,
WDRV_DISCONNECT_REASON_INVALID_PROFILE = 0x0A,
WDRV_DISCONNECT_REASON_PROFILE_MISMATCH = 0x0C,
WDRV_DISCONNECT_REASON_CONNECTION_EVICTED = 0x0d
```

Configuring the Library

The configuration of the MRF24WN Wi-Fi Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the Wi-Fi Driver. Based on the selections made, the MRF24WN Wi-Fi Driver may support the selected features. These configuration settings will apply to all instances of the MRF24WN Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Sample Functionality

The following code provides an example of Wi-Fi Driver configuration.

```
/*** Wi-Fi Driver Configuration ***/

#define WIFI_USE_RTOS

#define WDRV_EXT_INIT_TASK_STACK_SIZE 512u
#define WDRV_EXT_INIT_TASK_PRIO 6u
#define WDRV_EXT_MAIN_TASK_STACK_SIZE 2048u
#define WDRV_EXT_MAIN_TASK_PRIO 7u

#define WDRV_ASSERT(condition, msg) WDRV_Assert(condition, msg, __FILE__, __LINE__)

#define DRV_WIFI_SPI_INDEX 0
#define DRV_WIFI_SPI_INSTANCE sysObj.spiObjectIdx0

#define DRV_WIFI_NVM_SPACE_ENABLE
#define DRV_WIFI_NVM_SPACE_ADDR (48*1024)

#define MRF_INT_SOURCE INT_SOURCE_EXTERNAL_1
#define MRF_INT_VECTOR INT_VECTOR_INT1

// IO mapping for general control pins, including CS, RESET and HIBERNATE
// MRF24W in SPI 1 slot
#define WF_CS_PORT_CHANNEL PORT_CHANNEL_E
#define WF_CS_BIT_POS      9

#define WF_RESET_PORT_CHANNEL PORT_CHANNEL_F
#define WF_RESET_BIT_POS      0

#define WF_HIBERNATE_PORT_CHANNEL PORT_CHANNEL_F
#define WF_HIBERNATE_BIT_POS      1

#define WF_INT_PRIORITY     3
#define WF_INT_SUBPRIORITY  1
#define WF_INT_PORT_CHANNEL PORT_CHANNEL_E
#define WF_INT_BIT_POS      8

#define WDRV_DEFAULT_NETWORK_TYPE WDRV_NETWORK_TYPE_INFRASTRUCTURE
#define WDRV_DEFAULT_SSID_NAME "MicrochipDemoApp"

#define WDRV_DEFAULT_WIFI_SECURITY_MODE WDRV_SECURITY_OPEN
#define WDRV_DEFAULT_WEP_KEYS_40 "5AFB6C8E77" // default WEP40 key
#define WDRV_DEFAULT_WEP_KEYS_104 "90E96780C739409DA50034FCAA" // default WEP104 key
#define WDRV_DEFAULT_PSK_PHRASE "Microchip 802.11 Secret PSK Password" // default WPA-PSK or WPA2-PSK
passphrase
#define WDRV_DEFAULT_WPS_PIN "12390212" // default WPS PIN
```

```
#define WDRV_DEFAULT_CHANNEL 6
#define WDRV_DEFAULT_POWER_SAVE WDRV_FUNC_DISABLED
```

Building the Library

This section lists the files that are available in the MRF24WN Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/mrf24wn`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`wdrv_mrf24wn_common.h`	Contains all data types, define constants for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_api.h`	Contains function prototypes for interfacing to the MRF24WN Wi-Fi Driver.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`wdrv_mrf24wn_cli.c`	Provides access to MRF24WN Wi-Fi Driver controller.
`wdrv_mrf24wn_config_data.c`	Stores and retrieves MRF24WN Wi-Fi Driver configuration information in Non-volatile Memory (NVM).
`wdrv_mrf24wn_connmgr.c`	Provides access to MRF24WN Wi-Fi Driver controller for connection manager.
`wdrv_mrf24wn_events.c`	Provides access to MRF24WN Wi-Fi Driver controller for MAC events.
`wdrv_mrf24wn_iwpriv.c`	Provides functions to configure optional (private) parameters of the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_main.c`	Module for Microchip TCP/IP Stack PIC32 implementation for multiple Wi-Fi MAC support.
`wdrv_mrf24wn_misc.c`	Miscellaneous support functions and data types for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_osal.c`	RTOS wrapper functions for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_scan_helper.c`	Provides helper functions to access scan results.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	The MRF24WN Wi-Fi Driver controller has no optional files.

Module Dependencies

The MRF24WN Wi-Fi Driver Library depends on the following modules:

- SPI Driver Library
- NVM Driver Library
- UART Driver Library
- USB Driver Library
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library Introduction
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Debug System Service Library
- Ports System Service Library

- FreeRTOS Library Help
- Crypto Library
- Peripheral Libraries

Console Commands

This section describes the console commands available for the MRF24WN Wi-Fi Driver.

Description

Both the Web Server and the EasyConfig demonstrations support the followings commands, which enable control over the Wi-Fi settings.

Command: deleteconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.

[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example:
	`iwconfig security open`
	`iwconfig security wep40 <key>`
	`iwconfig security wep104 <key>`
	`iwconfig security wpa <key>`
	`iwconfig security wpa2 <key>`
	`iwconfig security pin <pin>`
	`iwconfig security pbc`

[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the MRF24WN module.

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Library Interface

a) Wi-Fi Initialization Functions

	Name	Description
	WDRV_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic

	WDRV_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic
	WDRV_GPIO_Init	Initializes the GPIO objects for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_PowerOff	Powers off the MRF24WN module. **Implementation:** Dynamic
	WDRV_GPIO_PowerOn	Powers on the MRF24WN module. **Implementation:** Dynamic
	WDRV_IsPowerOff	Checks if MRF24WN is turned off. **Implementation:** Dynamic
	WDRV_MRF24WN_ISR	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic

b) Wi-Fi Status Functions

	Name	Description
	WDRV_EXT_CmdConnectContextChannelGet	Gets the AP channel **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSaveGet	Retrieves current power save status. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the MRF24WN module. **Implementation:** Dynamic

c) External Functions

	Name	Description
	WDRV_EXT_CmdNetModeIBSSSet	Sets the Wi-Fi network type to Adhoc. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPA2Set	Sets Wi-Fi security to WPA2. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the WINC1500 Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_PrivConfig	Configures g_wdrvext_priv parameter. **Implementation:** Dynamic

e) Private Configuration Functions

	Name	Description
	iwpriv_config_write	Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_connstatus_get	Gets the Wi-Fi connection status. **Implementation:** Dynamic
	iwpriv_devinfo_get	Gets the device information. **Implementation:** Dynamic
	iwpriv_initialconn_set	Sets the initial connection status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_initstatus_get	Gets the initialization status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_is_servermode	Checks if the passed Wi-Fi context configuration is operating in server mode. **Implementation:** Dynamic
	iwpriv_leftclient_get	Gets the left client's information. **Implementation:** Dynamic
	iwpriv_mcastfilter_set	Adds a MAC address to the multi-cast filter. **Implementation:** Dynamic
	iwpriv_nettype_get	Gets the current network type. **Implementation:** Dynamic
	iwpriv_nettype_set	Sets the current network type. **Implementation:** Dynamic

	iwpriv_numberofscanresults_get	Gets the number of scan results. **Implementation:** Dynamic
	iwpriv_powersave_config	Enables or disables Power Save mode in Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_prescan_start	Starts prescan. **Implementation:** Dynamic
	iwpriv_scan_start	Starts scan. **Implementation:** Dynamic
	iwpriv_scanstatus_get	Gets the prescan status. **Implementation:** Dynamic
	iwpriv_ssid_get	Gets the current SSID. **Implementation:** Dynamic
	iwpriv_ssid_set	Sets the current SSID. **Implementation:** Dynamic
	iwpriv_execute	This is function iwpriv_execute.
	iwpriv_get	This is function iwpriv_get.
	iwpriv_prescan_isfinished	Checks if the prescan is complete. **Implementation:** Dynamic
	iwpriv_prescan_option_get	To see if prescan will run before next connection. **Implementation:** Dynamic
	iwpriv_prescan_option_set	To run prescan or not. **Implementation:** Dynamic
	iwpriv_set	This is function iwpriv_set.
	iwpriv_adhocctx_set	Sets the Ad hoc network context information. **Implementation:** Dynamic
	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	IWPRIV_CONN_STATUS	This is type IWPRIV_CONN_STATUS.
	IWPRIV_STATUS	This is type IWPRIV_STATUS.
	IWPRIV_CMD	This is type IWPRIV_CMD.
	IWPRIV_EXECUTE_PARAM	This is type IWPRIV_EXECUTE_PARAM.
	IWPRIV_GET_PARAM	This is type IWPRIV_GET_PARAM.
	IWPRIV_PARAM_CLIENTINFO	This is type IWPRIV_PARAM_CLIENTINFO.
	IWPRIV_PARAM_CONTEXT	This is type IWPRIV_PARAM_CONTEXT.
	IWPRIV_PARAM_DEVICEINFO	This is type IWPRIV_PARAM_DEVICEINFO.
	IWPRIV_SCAN_STATUS	This is type IWPRIV_SCAN_STATUS.
	IWPRIV_SET_PARAM	This is type IWPRIV_SET_PARAM.
	IWPRIV_PARAM_CONFIG	This is type IWPRIV_PARAM_CONFIG.
	IWPRIV_PARAM_CONNECT	This is type IWPRIV_PARAM_CONNECT.
	IWPRIV_PARAM_DRIVERSTATUS	This is type IWPRIV_PARAM_DRIVERSTATUS.
	IWPRIV_PARAM_FWUPGRADE	This is type IWPRIV_PARAM_FWUPGRADE.
	IWPRIV_PARAM_MULTICASTFILTER	This is type IWPRIV_PARAM_MULTICASTFILTER.
	IWPRIV_PARAM_NETWORKTYPE	This is type IWPRIV_PARAM_NETWORKTYPE.
	IWPRIV_PARAM_OPERATIONMODE	This is type IWPRIV_PARAM_OPERATIONMODE.
	IWPRIV_PARAM_POWERSAVE	This is type IWPRIV_PARAM_POWERSAVE.
	IWPRIV_PARAM_SCAN	This is type IWPRIV_PARAM_SCAN.
	IWPRIV_PARAM_SSID	This is type IWPRIV_PARAM_SSID.

Description

This section describes the Application Programming Interface (API) functions of the MRF24WN Wi-Fi Driver.

Refer to each section for a detailed description.

a) Wi-Fi Initialization Functions

WDRV_SPI_In Function

Receives data from the module through the SPI bus.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_In(uint8_t *const OutBuf, uint16_t OutSize, uint8_t *const InBuf, uint16_t InSize);
```

Returns

None.

Description

This function receives data from the module through the SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
bufOut	buffer pointer of output command
OutSize	the command size
InBuf	buffer pointer of input data
InSize	the input data size

Function

void WDRV_SPI_In(uint8_t const *const OutBuf, uint16_t OutSize,
uint8_t *const InBuf, uint16_t InSize)

WDRV_SPI_Out Function

Sends data out to the module through the SPI bus.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Out(uint8_t *const bufOut, uint16_t OutSize);
```

Returns

None.

Description

This function sends data out to the module through the SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
bufOut	buffer pointer of output data

OutSize	the data size

Function

void WDRV_SPI_Out(uint8_t const *const bufOut, uint16_t OutSize)

WDRV_GPIO_Init Function

Initializes the GPIO objects for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_Init();
```

Returns

None.

Description

This function initializes the GPIO objects for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_Init(void)

WDRV_GPIO_PowerOff Function

Powers off the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_PowerOff();
```

Returns

None.

Description

This function powers off the MRF24WN module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_PowerOff(void)

WDRV_GPIO_PowerOn Function

Powers on the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_PowerOn();
```

Returns

None.

Description

This function powers on the MRF24WN module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_PowerOn(void)

WDRV_IsPowerOff Function

Checks if MRF24WN is turned off.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
bool WDRV_IsPowerOff();
```

Returns

- 0 - Indicates that MRF24WN is turned off
- Non-zero value - Indicates that MRF24WN is on

Description

This function checks if MRF24WN is turned off.

Remarks

None.

Function

bool WDRV_IsPowerOff(void)

WDRV_MRF24WN_ISR Function

Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_MRF24WN_ISR();
```

Returns

None.

Description

This function is the Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_MRF24WN_ISR(void)

b) Wi-Fi Status Functions

WDRV_EXT_CmdConnectContextChannelGet Function

Gets the AP channel

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdConnectContextChannelGet(uint16_t * bssChannel);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function gets the current AP channel.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
bssChannel	pointer where the current AP channel will be written

Function

uint32_t WDRV_EXT_CmdConnectContextChannelGet(uint16_t *bssChannel)

WDRV_EXT_CmdPowerSaveGet Function

Retrieves current power save status.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdPowerSaveGet(bool * enabled);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function retrieves the current power save status.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enabled	pointer where the current power save status will be written

Function

uint32_t WDRV_EXT_CmdPowerSaveGet(bool *enabled)

WDRV_EXT_ScanResultGet Function

Reads the selected scan results back from the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_ScanResultGet(uint16_t idx, WDRV_SCAN_RESULT * p_scanResult);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

After a scan has completed this function is used to read one scan result at a time from the MRF24WN module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
listIndex	index (0 based list) of the scan entry to retrieve
p_scanResult	pointer to where scan result is written

Function

void WDRV_EXT_ScanResultGet(uint8_t listIndex, WDRV_SCAN_RESULT *p_scanResult)

c) External Functions

WDRV_EXT_CmdNetModeIBSSSet Function

Sets the Wi-Fi network type to Adhoc.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdNetModeIBSSSet();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi network type to Adhoc.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdNetModeIBSSSet(void)

WDRV_EXT_CmdSecWPA2Set Function

Sets Wi-Fi security to WPA2.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecWPA2Set(uint8_t * key, uint16_t len);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WPA2. One can only connect to an AP that is running the same WPA2 mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WPA2 key buffer
len	WPA2 key length

Function

uint32_t WDRV_EXT_CmdSecWPA2Set(uint8_t *key, uint16_t len)

WDRV_EXT_Initialize Function

Initializes the MRF24WN Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
int32_t WDRV_EXT_Initialize(const WDRV_CALLBACKS *const CB);
```

Returns

- 0 - Indicates success

- non-zero value - Indicates failure

Description

This function initializes the MRF24WN Wi-Fi driver, making it ready for clients to use.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
CB	pointer to callback functions

Function

int32_t WDRV_EXT_Initialize(const WDRV_CALLBACKS *const CB)

WDRV_EXT_Initialize Function

Initializes the WINC1500 Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_Initialize(const WDRV_HOOKS *const ehooks, bool initWait);
```

Returns

None.

Description

This function initializes the WINC1500 Wi-Fi driver, making it ready for clients to use.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
ehooks	pointer to WDRV layer hooks
initWait	true will put WDRV in wait during initialization

Function

void WDRV_EXT_Initialize(const WDRV_HOOKS *const ehooks, bool initWait)

WDRV_EXT_PrivConfig Function

Configures g_wdrvext_priv parameter.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_PrivConfig(uint32_t * config);
```

Returns

None.

Description

This function configures g_wdrvext_priv parameter.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
config	pointer to the parameter array

Function

void WDRV_EXT_PrivConfig(uint32_t *config)

d) GPIO Functions

e) Private Configuration Functions

iwpriv_config_write Function

Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_config_write(void * wifi_cfg);
```

Returns

None.

Description

This function reads from a passed pointer, copies everything from it, and writes to the Wi-Fi context configuration, which is currently used by the Wi-Fi driver.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
wifi_cfg	pointer to where the context configuration is stored

Function

void iwpriv_config_write(void *wifi_cfg)

iwpriv_connstatus_get Function

Gets the Wi-Fi connection status.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_CONN_STATUS iwpriv_connstatus_get();
```

Returns

Status of current Wi-Fi connection. See the definition for the IWPRIV_CONN_STATUS structure.

Description

This function gets the Wi-Fi connection status.

Remarks

IWPRIV_CONNECTION_FAILED does not necessarily mean that the module fails to connect to the network. It stands on the application's perspective, and actually can be customized. For example, in the Web Server demonstrations's use case, WDRV_CSTATE_CONNECTION_PERMANENTLY_LOST is treated as a fail case and will trigger the application to restart.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_CONN_STATUS iwpriv_connstatus_get(void)

iwpriv_devinfo_get Function

Gets the device information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_devinfo_get(void * info);
```

Returns

None.

Description

This function returns the device information.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
info	pointer to where the device information is written

Function

void iwpriv_devinfo_get(void *info)

iwpriv_initialconn_set Function

Sets the initial connection status of Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_initialconn_set(bool connect);
```

Returns

None.

Description

This function sets the initial connection status of Wi-Fi driver. After Wi-Fi initialization, it decides whether or not to start the Wi-Fi connection.

Remarks

This function is mainly used to implement prescan. It has to be called before Wi-Fi driver's initialization is finished to be effective.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
connect	boolean value which indicates whether or not to start an initial connect

Function

void iwpriv_initialconn_set(bool connect)

iwpriv_initstatus_get Function

Gets the initialization status of Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
uint8_t iwpriv_initstatus_get();
```

Returns

Current initialization status of the Wi-Fi driver (IWPRIV_READY or IWPRIV_IN_PROGRESS).

Description

This function returns the initialization status of the Wi-Fi driver.

Remarks

None.

Preconditions

None.

Function

uint8_t iwpriv_initstatus_get(void)

iwpriv_is_servermode Function

Checks if the passed Wi-Fi context configuration is operating in server mode.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_is_servermode();
```

Returns

- true - Wi-Fi context configuration is operating in server mode

- false - Wi-Fi context configuration is not operating in server mode

Description

This function checks if the passed Wi-Fi context configuration is operating in server mode, which includes Ad hoc mode and SoftAP mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_is_servermode(void)

iwpriv_leftclient_get Function

Gets the left client's information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_leftclient_get(bool * updated, TCPIP_MAC_ADDR * addr);
```

Returns

None.

Description

This function returns the left client's information when the Wi-Fi module works in server mode and has the DHCP Server enabled.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
updated	if the left client's information needs to be updated
addr	MAC address of the left client

Function

void iwpriv_leftclient_get(bool *updated, TCPIP_MAC_ADDR *addr)

iwpriv_mcastfilter_set Function

Adds a MAC address to the multi-cast filter.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_STATUS iwpriv_mcastfilter_set(uint8_t * addr);
```

Returns

Status of the set operation, IWPRIV_READY or IWPRIV_ERROR. See definition for the IWPRIV_STATUS structure.

Description

This function adds a MAC address to the multi-cast filter.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
addr	pointer to where the MAC address is stored

Function

IWPRIV_STATUS iwpriv_mcastfilter_set(uint8_t *addr)

iwpriv_nettype_get Function

Gets the current network type.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_nettype_get(uint8_t * netType);
```

Returns

None.

Description

This function returns the current network type.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
netType	pointer to where the network type is written

Function

void iwpriv_nettype_get(uint8_t *netType)

iwpriv_nettype_set Function

Sets the current network type.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_nettype_set(uint8_t netType);
```

Returns

None.

Description

This function sets the current network type.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
netType	the network type to set

Function

void iwpriv_nettype_set(uint8_t netType)

iwpriv_numberofscanresults_get Function

Gets the number of scan results.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
uint16_t iwpriv_numberofscanresults_get();
```

Returns

Number of scan results.

Description

This function gets the number of scan results.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint16_t iwpriv_numberofscanresults_get(void)

iwpriv_powersave_config Function

Enables or disables Power Save mode in Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_powersave_config(bool enabled);
```

Returns

None.

Description

This function enables or disables Power Save mode in Wi-Fi driver, which depends on the passed boolean value.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enabled	boolean value which indicates to enable or disable Power Save mode in Wi-Fi driver

Function

void iwpriv_powersave_config(bool enabled)

iwpriv_prescan_start Function

Starts prescan.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_prescan_start();
```

Returns

None.

Description

This function directs the Wi-Fi driver to start a prescan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void iwpriv_prescan_start(void)

iwpriv_scan_start Function

Starts scan.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_scan_start();
```

Returns

None.

Description

The function starts a Wi-Fi scan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void iwpriv_scan_start(void)

iwpriv_scanstatus_get Function

Gets the prescan status.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_SCAN_STATUS iwpriv_scanstatus_get();
```

Returns

Prescan status: IWPRIV_SCAN_IDLE, IWPRIV_SCAN_IN_PROGRESS, IWPRIV_SCAN_NO_AP_FOUND or IWPRIV_SCAN_SUCCESSFUL. See the definition for the IWPRIV_SCAN_STATUS structure.

Description

This function gets the prescan status.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_SCAN_STATUS iwpriv_scanstatus_get(void)

iwpriv_ssid_get Function

Gets the current SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_ssid_get(uint8_t * ssid, uint8_t * ssidLen);
```

Returns

None.

Description

This function returns the current SSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to where the SSID is written
ssidLen	pointer to where the SSID length is written

Function

void iwpriv_ssid_get(uint8_t *ssid, uint8_t *ssidLen)

iwpriv_ssid_set Function

Sets the current SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_ssid_set(uint8_t * ssid, uint8_t ssidLen);
```

Returns

None.

Description

This function sets the current SSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to where the SSID is stored
ssidLen	pointer to where the SSID length is stored

Function

void iwpriv_ssid_set(uint8_t *ssid, uint8_t ssidLen)

iwpriv_execute Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_execute(IWPRIV_CMD cmd, IWPRIV_EXECUTE_PARAM * params);
```

Description

This is function iwpriv_execute.

iwpriv_get Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_get(IWPRIV_CMD cmd, IWPRIV_GET_PARAM * params);
```

Description

This is function iwpriv_get.

iwpriv_prescan_isfinished Function

Checks if the prescan is complete.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_prescan_isfinished();
```

Returns

None.

Description

This function checks if the prescan is complete.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_prescan_isfinished(void)

iwpriv_prescan_option_get Function

To see if prescan will run before next connection.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_prescan_option_get();
```

Returns

None.

Description

This function checks whether or not the prescan will run before next connection.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_prescan_option_get(void)

iwpriv_prescan_option_set Function

To run prescan or not.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_prescan_option_set(bool scan);
```

Returns

None.

Description

This function controls whether or not to run prescan.

Remarks

Prescan means the scan runs before the module is connected. It needs to use multiple functions in this file. Please refer to the Easy Configuration demonstration to see the correct usage of prescan.

After the the module is connected, MRF24WN module can also do regular scans. But it cannot perform a scan when the connection is in progress.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
scan	true: run prescan before next connection
false	do not run prescan before next connection

Function

void iwpriv_prescan_option_set(bool scan)

iwpriv_set Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_set(IWPRIV_CMD cmd, IWPRIV_SET_PARAM * params);
```

Description

This is function iwpriv_set.

iwpriv_adhocctx_set Function

Sets the Ad hoc network context information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_adhocctx_set(void * p_cxt);
```

Returns

None.

Description

This function sets the current Ad hoc network context information by reading from a passed pointer.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
p_cxt	pointer to where the Ad hoc network context is stored

Function

void iwpriv_adhocctx_set(void *p_cxt)

iwpriv_config_read Function

Reads the Wi-Fi context configuration.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_config_read(void * wifi_cfg);
```

Returns

None.

Description

This function reads the current Wi-Fi context configuration, copies and stores the whole structure to the pointer passed to the function.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
wifi_cfg	pointer to where the context configuration is written

Function

void iwpriv_config_read(void *wifi_cfg)

f) Data Types and Constants

IWPRIV_CONN_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_CONNECTION_FAILED = -1,
  IWPRIV_CONNECTION_SUCCESSFUL,
  IWPRIV_CONNECTION_IDLE,
  IWPRIV_CONNECTION_IN_PROGRESS,
  IWPRIV_CONNECTION_REESTABLISHED
} IWPRIV_CONN_STATUS;
```

Description

This is type IWPRIV_CONN_STATUS.

IWPRIV_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_ERROR = -1,
  IWPRIV_READY,
  IWPRIV_IN_PROGRESS
} IWPRIV_STATUS;
```

Description

This is type IWPRIV_STATUS.

IWPRIV_CMD Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  PRESCAN_OPTION_GET,
  PRESCAN_OPTION_SET,
  PRESCAN_START,
  PRESCAN_ISFINISHED_GET,
  SCAN_START,
  SCANSTATUS_GET,
  SCANRESULT_GET,
  SCANRESULTS_COUNT_GET,
  CONFIG_GET,
  CONFIG_SET,
  SSID_GET,
  SSID_SET,
  NETWORKTYPE_GET,
  NETWORKTYPE_SET,
  CONNSTATUS_GET,
  CLIENTINFO_GET,
  DEVICEINFO_GET,
  DRVSTATUS_GET,
  FWUPGRADEREQUEST_GET,
  OPERATIONMODE_GET,
  INITCONN_OPTION_SET,
  ADHOCCTX_SET,
  MULTICASTFILTER_SET,
  POWERSAVE_SET
} IWPRIV_CMD;
```

Description

This is type IWPRIV_CMD.

IWPRIV_EXECUTE_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
} IWPRIV_EXECUTE_PARAM;
```

Description

This is type IWPRIV_EXECUTE_PARAM.

IWPRIV_GET_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
  IWPRIV_PARAM_SCAN scan;
  IWPRIV_PARAM_CONFIG cfg;
  IWPRIV_PARAM_SSID ssid;
  IWPRIV_PARAM_NETWORKTYPE netType;
  IWPRIV_PARAM_CONNECT conn;
  IWPRIV_PARAM_CLIENTINFO clientInfo;
  IWPRIV_PARAM_DEVICEINFO devInfo;
```

```
  IWPRIV_PARAM_DRIVERSTATUS driverStatus;
  IWPRIV_PARAM_FWUPGRADE fwUpgrade;
  IWPRIV_PARAM_OPERATIONMODE opMode;
} IWPRIV_GET_PARAM;
```

Description

This is type IWPRIV_GET_PARAM.

IWPRIV_PARAM_CLIENTINFO Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * addr;
  bool updated;
} IWPRIV_PARAM_CLIENTINFO;
```

Members

Members	Description
uint8_t * addr;	it usually points to a MAC address, which is an array of 6 uint8_t elements

Description

This is type IWPRIV_PARAM_CLIENTINFO.

IWPRIV_PARAM_CONTEXT Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * context;
} IWPRIV_PARAM_CONTEXT;
```

Description

This is type IWPRIV_PARAM_CONTEXT.

IWPRIV_PARAM_DEVICEINFO Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * info;
} IWPRIV_PARAM_DEVICEINFO;
```

Description

This is type IWPRIV_PARAM_DEVICEINFO.

IWPRIV_SCAN_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_SCAN_SUCCESSFUL,
  IWPRIV_SCAN_IDLE,
  IWPRIV_SCAN_IN_PROGRESS,
```

```
    IWPRIV_SCAN_NO_AP_FOUND
} IWPRIV_SCAN_STATUS;
```

Description

This is type IWPRIV_SCAN_STATUS.

IWPRIV_SET_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
  IWPRIV_PARAM_SCAN scan;
  IWPRIV_PARAM_CONFIG cfg;
  IWPRIV_PARAM_SSID ssid;
  IWPRIV_PARAM_NETWORKTYPE netType;
  IWPRIV_PARAM_CONNECT conn;
  IWPRIV_PARAM_CONTEXT ctx;
  IWPRIV_PARAM_MULTICASTFILTER multicast;
  IWPRIV_PARAM_POWERSAVE powerSave;
} IWPRIV_SET_PARAM;
```

Description

This is type IWPRIV_SET_PARAM.

IWPRIV_PARAM_CONFIG Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * config;
} IWPRIV_PARAM_CONFIG;
```

Description

This is type IWPRIV_PARAM_CONFIG.

IWPRIV_PARAM_CONNECT Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool initConnAllowed;
  IWPRIV_CONN_STATUS status;
} IWPRIV_PARAM_CONNECT;
```

Description

This is type IWPRIV_PARAM_CONNECT.

IWPRIV_PARAM_DRIVERSTATUS Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool isOpen;
} IWPRIV_PARAM_DRIVERSTATUS;
```

Description

This is type IWPRIV_PARAM_DRIVERSTATUS.

IWPRIV_PARAM_FWUPGRADE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool requested;
} IWPRIV_PARAM_FWUPGRADE;
```

Description

This is type IWPRIV_PARAM_FWUPGRADE.

IWPRIV_PARAM_MULTICASTFILTER Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * addr;
  IWPRIV_STATUS status;
} IWPRIV_PARAM_MULTICASTFILTER;
```

Members

Members	Description
uint8_t * addr;	it usually points to a MAC address, which is an array of 6 uint8_t elements

Description

This is type IWPRIV_PARAM_MULTICASTFILTER.

IWPRIV_PARAM_NETWORKTYPE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t type;
} IWPRIV_PARAM_NETWORKTYPE;
```

Description

This is type IWPRIV_PARAM_NETWORKTYPE.

IWPRIV_PARAM_OPERATIONMODE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool isServer;
} IWPRIV_PARAM_OPERATIONMODE;
```

Description

This is type IWPRIV_PARAM_OPERATIONMODE.

IWPRIV_PARAM_POWERSAVE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool enabled;
} IWPRIV_PARAM_POWERSAVE;
```

Description

This is type IWPRIV_PARAM_POWERSAVE.

IWPRIV_PARAM_SCAN Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool prescanAllowed;
  bool prescanFinished;
  IWPRIV_SCAN_STATUS scanStatus;
  uint16_t numberOfResults;
  uint16_t index;
  WDRV_SCAN_RESULT * result;
} IWPRIV_PARAM_SCAN;
```

Description

This is type IWPRIV_PARAM_SCAN.

IWPRIV_PARAM_SSID Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * ssid;
  uint8_t ssidLen;
} IWPRIV_PARAM_SSID;
```

Description

This is type IWPRIV_PARAM_SSID.

Files

Files

Name	Description
wdrv_mrf24wn_api.h	MRF24WN Interface Functions
wdrv_mrf24wn_iwpriv.h	Configure optional (private) parameters of MRF24WN driver.

Description

This section lists the source and header files used by the MRF24WN Wi-Fi Driver Library.

wdrv_mrf24wn_api.h

MRF24WN Interface Functions

Functions

	Name	Description
	WDRV_EXT_CmdConnectContextChannelGet	Gets the AP channel **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeIBSSSet	Sets the Wi-Fi network type to Adhoc. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSaveGet	Retrieves current power save status. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPA2Set	Sets Wi-Fi security to WPA2. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_PrivConfig	Configures g_wdrvext_priv parameter. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the MRF24WN module. **Implementation:** Dynamic
	WDRV_GPIO_DeInit	Deinitializes the GPIO objects for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_Init	Initializes the GPIO objects for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_PowerOff	Powers off the MRF24WN module. **Implementation:** Dynamic
	WDRV_GPIO_PowerOn	Powers on the MRF24WN module. **Implementation:** Dynamic
	WDRV_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic
	WDRV_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_IsPowerOff	Checks if MRF24WN is turned off. **Implementation:** Dynamic
	WDRV_MRF24WN_ISR	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_SPI_Deinit	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic
	WDRV_SPI_Init	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic

Description

MRF24WN Interface Functions

File Name

wdrv_mrf24wn_api.h

Company

Microchip Technology Inc.

wdrv_mrf24wn_iwpriv.h

Configure optional (private) parameters of MRF24WN driver.

Enumerations

	Name	Description
	IWPRIV_CMD	This is type IWPRIV_CMD.
	IWPRIV_CONN_STATUS	This is type IWPRIV_CONN_STATUS.
	IWPRIV_SCAN_STATUS	This is type IWPRIV_SCAN_STATUS.
	IWPRIV_STATUS	This is type IWPRIV_STATUS.

Functions

	Name	Description
	iwpriv_adhocctx_set	Sets the Ad hoc network context information. **Implementation:** Dynamic
	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic
	iwpriv_config_write	Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_connstatus_get	Gets the Wi-Fi connection status. **Implementation:** Dynamic
	iwpriv_devinfo_get	Gets the device information. **Implementation:** Dynamic
	iwpriv_execute	This is function iwpriv_execute.
	iwpriv_get	This is function iwpriv_get.
	iwpriv_initialconn_set	Sets the initial connection status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_initstatus_get	Gets the initialization status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_is_servermode	Checks if the passed Wi-Fi context configuration is operating in server mode. **Implementation:** Dynamic
	iwpriv_leftclient_get	Gets the left client's information. **Implementation:** Dynamic
	iwpriv_mcastfilter_set	Adds a MAC address to the multi-cast filter. **Implementation:** Dynamic
	iwpriv_nettype_get	Gets the current network type. **Implementation:** Dynamic
	iwpriv_nettype_set	Sets the current network type. **Implementation:** Dynamic
	iwpriv_numberofscanresults_get	Gets the number of scan results. **Implementation:** Dynamic
	iwpriv_powersave_config	Enables or disables Power Save mode in Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_prescan_isfinished	Checks if the prescan is complete. **Implementation:** Dynamic
	iwpriv_prescan_option_get	To see if prescan will run before next connection. **Implementation:** Dynamic
	iwpriv_prescan_option_set	To run prescan or not. **Implementation:** Dynamic
	iwpriv_prescan_start	Starts prescan. **Implementation:** Dynamic
	iwpriv_scan_start	Starts scan. **Implementation:** Dynamic
	iwpriv_scanstatus_get	Gets the prescan status. **Implementation:** Dynamic
	iwpriv_set	This is function iwpriv_set.
	iwpriv_ssid_get	Gets the current SSID. **Implementation:** Dynamic
	iwpriv_ssid_set	Sets the current SSID. **Implementation:** Dynamic

Structures

	Name	Description
	IWPRIV_PARAM_CLIENTINFO	This is type IWPRIV_PARAM_CLIENTINFO.
	IWPRIV_PARAM_CONFIG	This is type IWPRIV_PARAM_CONFIG.
	IWPRIV_PARAM_CONNECT	This is type IWPRIV_PARAM_CONNECT.
	IWPRIV_PARAM_CONTEXT	This is type IWPRIV_PARAM_CONTEXT.
	IWPRIV_PARAM_DEVICEINFO	This is type IWPRIV_PARAM_DEVICEINFO.
	IWPRIV_PARAM_DRIVERSTATUS	This is type IWPRIV_PARAM_DRIVERSTATUS.
	IWPRIV_PARAM_FWUPGRADE	This is type IWPRIV_PARAM_FWUPGRADE.
	IWPRIV_PARAM_MULTICASTFILTER	This is type IWPRIV_PARAM_MULTICASTFILTER.
	IWPRIV_PARAM_NETWORKTYPE	This is type IWPRIV_PARAM_NETWORKTYPE.
	IWPRIV_PARAM_OPERATIONMODE	This is type IWPRIV_PARAM_OPERATIONMODE.
	IWPRIV_PARAM_POWERSAVE	This is type IWPRIV_PARAM_POWERSAVE.
	IWPRIV_PARAM_SCAN	This is type IWPRIV_PARAM_SCAN.
	IWPRIV_PARAM_SSID	This is type IWPRIV_PARAM_SSID.

Unions

	Name	Description
	IWPRIV_EXECUTE_PARAM	This is type IWPRIV_EXECUTE_PARAM.
	IWPRIV_GET_PARAM	This is type IWPRIV_GET_PARAM.
	IWPRIV_SET_PARAM	This is type IWPRIV_SET_PARAM.

Description

MRF24WN Private Configuration Support

Functions in this module support the connection process for the MRF24WN.

File Name

wdrv_mrf24wn_iwpriv.h

Company

Microchip Technology Inc.

WILC1000 Wi-Fi Driver Ethernet Mode Library

This topic describes the WILC1000 Wi-Fi Driver Library.

Introduction

This library provides a low-level abstraction of the WILC1000 Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Note: The WILC1000 Wi-Fi Driver is compatible with the WILC1000 PICtail/PICtail Plus Daughter Board with WILC1000 firmware version 4.2.3 and later in "Ethernet mode".

Description

The Wi-Fi software library, in conjunction with the WILC1000 module, allows an application to:

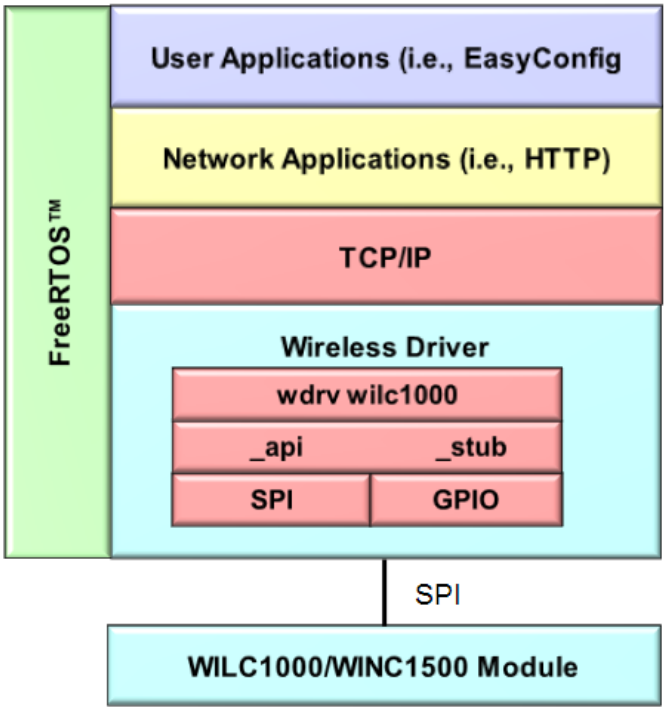
- Join an existing 802.11 Wi-Fi network
- Create a 802.11 Wi-Fi network

The following application services are provided by the Wi-Fi library:

- Configure a Wi-Fi connection (SSID, security mode, and so on)
- Join an existing network or create a "Soft-AP" Wi-Fi network
- Scan for other Wi-Fi devices in the area
- Receive Wi-Fi network status
- Wi-Fi power control

The MAC_layer services are not directly accessible to the application. This portion of the code resides under the TCP/IP Stack MAC module software layers and is used by stack services to transmit and receive data over a Wi-Fi network. The following diagram shows the interaction of the primary software blocks in a Wi-Fi application.

Wi-Fi Software Block Diagram



The following table provides information that includes network mode and security mode support by the WILC1000 Wi-Fi Driver.

WILC1000 Network Connection Matrix	Network Mode	
Security Mode	Infrastructure	SoftAP
Open	YES	YES
WEP40 (64-bit)	YES	YES
WEP104 (128-bit)	YES	YES
WPA-AUTO PSK (TKIP/AES)	YES	NA
WPA-AUTO PSK (AES)	YES	YES
WPS Push Button	YES	NA
WPS PIN	YES	NA

Using the Library

This topic describes the basic architecture of the WILC1000 Wi-Fi Driver Library and provides information and examples on its use.

Description

Interface Header Files: `wdrv_wilc1000_api.h` and `wdrv_wilc1000_stub.h`

The interface to the WILC1000 Wi-Fi Driver Library is defined in the `wdrv_wilc1000_api.h` and `wdrv_wilc1000_stub.h` header files.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the WILC1000 Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The WILC1000 Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing

- Wi-Fi status
- Wi-FI console commands

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

TheLibrary Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the Wi-Fi module.

Library Interface Section	Description
Wi-Fi Initialization Functions	This section provides functions that initialize the Wi-Fi library and allow its API to be used.
Wi-Fi Status Functions	This section provides functions that retrieve the Wi-Fi connection status.
Wi-Fi External Functions	This section provides public functions accessible to TCP/IP applications.
Other Functions	This section provides additional miscellaneous functions for configuring the Wi-Fi connection.
Data Types and Constants	This section provides data types and macros.

How the Library Works

This section describes how the WILC1000 Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, it should be configured (compile time configuration).

There are a few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the WILC1000 Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

Configuring the Library

This section describes how to configure the WILC1000 Wi-Fi driver.

Description

The configuration of the WILC1000 Wi-Fi Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the Wi-Fi Driver. Based on the selections made, the WILC1000 Wi-Fi Driver may support the selected features. These configuration settings will apply to all instances of the WILC1000 Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Sample Functionality

The following code provides an example of Wi-Fi Driver configuration. (refer to `system.config.h`)

```
/*** SPI Driver Configuration ***/
#define DRV_SPI_NUMBER_OF_MODULES    4
/*** Driver Compilation and static configuration options. ***/
/*** Select SPI compilation units.***/
#define DRV_SPI_POLLED    0
#define DRV_SPI_ISR    1
#define DRV_SPI_MASTER    1
#define DRV_SPI_SLAVE    0
#define DRV_SPI_RM    1
#define DRV_SPI_EBM    0
#define DRV_SPI_8BIT    1
#define DRV_SPI_16BIT    0
#define DRV_SPI_32BIT    0
#define DRV_SPI_DMA    1
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER    1
#define DRV_SPI_CLIENTS_NUMBER    1
#define DRV_SPI_ELEMENTS_PER_QUEUE    10
/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE    512
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE    512
```

```
/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0    SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0    DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0    DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0    false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0    DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0    SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0    CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0    2000000
#define DRV_SPI_BUFFER_TYPE_IDX0    DRV_SPI_BUFFER_TYPE_STANDARD
#define DRV_SPI_CLOCK_MODE_IDX0    DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0    SPI_INPUT_SAMPLING_PHASE_AT_END


#define DRV_SPI_TRANSMIT_DUMMY_BYTE_VALUE_IDX0    0x00

#define DRV_SPI_TX_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0    INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0    INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0    INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0    10
#define DRV_SPI_RESERVED_JOB_IDX0    1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0    DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0    16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0    DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0    16
/*** Timer Driver Configuration ***/
#define DRV_TMR_INTERRUPT_MODE    true
#define DRV_TMR_INSTANCES_NUMBER    1
#define DRV_TMR_CLIENTS_NUMBER    1

/*** Timer Driver 0 Configuration ***/
#define DRV_TMR_PERIPHERAL_ID_IDX0    TMR_ID_2
#define DRV_TMR_INTERRUPT_SOURCE_IDX0    INT_SOURCE_TIMER_2
#define DRV_TMR_INTERRUPT_VECTOR_IDX0    INT_VECTOR_T2
#define DRV_TMR_ISR_VECTOR_IDX0    _TIMER_2_VECTOR
#define DRV_TMR_INTERRUPT_PRIORITY_IDX0    INT_PRIORITY_LEVEL4
#define DRV_TMR_INTERRUPT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_TMR_CLOCK_SOURCE_IDX0    DRV_TMR_CLKSOURCE_INTERNAL
#define DRV_TMR_PRESCALE_IDX0    TMR_PRESCALE_VALUE_256
#define DRV_TMR_OPERATION_MODE_IDX0    DRV_TMR_OPERATION_MODE_16_BIT
#define DRV_TMR_ASYNC_WRITE_ENABLE_IDX0    false
#define DRV_TMR_POWER_STATE_IDX0    SYS_MODULE_POWER_RUN_FULL

/*** Wi-Fi Driver Configuration ***/
#define WILC1000_INT_SOURCE INT_SOURCE_CHANGE_NOTICE
#define WILC1000_INT_VECTOR INT_VECTOR_CN

#define WDRV_SPI_INDEX 0
#define WDRV_SPI_INSTANCE sysObj.spiObjectIdx0

#define WDRV_USE_SPI_DMA

#define WDRV_NVM_SPACE_ENABLE
#define WDRV_NVM_SPACE_ADDR (48 * 1024)

#define WDRV_BOARD_TYPE WDRV_BD_TYPE_MX_ESK

#define WDRV_EXT_RTOS_TASK_SIZE 2048u
#define WDRV_EXT_RTOS_TASK_PRIORITY 2u

// I/O mappings for general control pins, including CHIP_EN, IRQN, RESET_N and SPI_SSN.
#define WDRV_CHIP_EN_PORT_CHANNEL    PORT_CHANNEL_F
#define WDRV_CHIP_EN_BIT_POS    1

#define WDRV_IRQN_PORT_CHANNEL    PORT_CHANNEL_G
```

```
#define WDRV_IRQN_BIT_POS    7

#define WDRV_RESET_N_PORT_CHANNEL    PORT_CHANNEL_F
#define WDRV_RESET_N_BIT_POS    0

#define WDRV_SPI_SSN_PORT_CHANNEL    PORT_CHANNEL_B
#define WDRV_SPI_SSN_BIT_POS    2

#define WILC1000_ON_PIC32MX_ESK

// On PIC32MX ESK, when CN9 (Pin G7) is used as external interrupt,
// it is sometimes better to use another GPIO (Pin E0) to read CN9's value.


// In this case, a jumper wire is needed to connect Pin E0 and Pin G7.
//#define WDRV_VERIFY_IRQN_BY_ANOTHER_GPIO
#if defined(WDRV_VERIFY_IRQN_BY_ANOTHER_GPIO)
// Use Pin E0. Please also make sure that Pin E0 and Pin G7 are connected (by a jumper wire).
#define WDRV_IRQN_PORT_CHANNEL_READ    PORT_CHANNEL_E
#define WDRV_IRQN_BIT_POS_READ    0
#else
// Still directly read Pin G7's value.
#define WDRV_IRQN_PORT_CHANNEL_READ    PORT_CHANNEL_G
#define WDRV_IRQN_BIT_POS_READ    7
#endif

#define WDRV_DEFAULT_NETWORK_TYPE WDRV_NETWORK_TYPE_INFRASTRUCTURE
#define WDRV_DEFAULT_CHANNEL 6
#define WDRV_DEFAULT_SSID "MicrochipDemoApp"

#define WDRV_DEFAULT_SECURITY_MODE WDRV_SECURITY_OPEN
#define WDRV_DEFAULT_WEP_KEYS_40 "5AFB6C8E77" // default WEP40 key
#define WDRV_DEFAULT_WEP_KEYS_104 "90E96780C739409DA50034FCAA" // default WEP104 key
#define WDRV_DEFAULT_PSK_PHRASE "Microchip 802.11 Secret PSK Password" // default WPA-PSK
```

Building the Library

This section lists the files that are available in the WILC1000 Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/wilc1000`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
wdrv_wilc1000_stub.h	Contains Stub function prototypes for interfacing to the WILC1000 Wi-Fi Driver.
wdrv_wilc1000_api.h	Contains API function prototypes for interfacing to the WILC1000 Wi-Fi Driver.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
wdrv_wilc1000_console.c	Console module for the WILC1000 wireless driver.
wdrv_wilc1000_fw_update.c	WILC1000 firmware update support.
wdrv_wilc1000_eint.c	External interrupt handler for the WILC1000 wireless driver.
wdrv_wilc1000_timer.c	Timer functions for the WILC1000 wireless driver.
wdrv_wilc1000_gpio.c	WILC1000 GPIO support for SPI communication.

wdrv_wilc1000_spi.c	WILC1000 support for SPI communication.
wdrv_wilc1000_cli.c	WILC1000 driver CLI implementation.
wdrv_wilc1000_config_data.c	Stores and retrieves Wi-Fi configuration to/from non-volatile memory (NVM).
wdrv_wilc1000_connmgr.c	WILC1000 driver connection manager.
wdrv_wilc1000_events.c	WILC1000 driver MAC events.
wdrv_wilc1000_iwpriv.c	WILC1000 driver connection process functions.
wdrv_wilc1000_main.c	WILC1000 driver Microchip TCP/IP Stack PIC32 MAC support.
wdrv_wilc1000_osal.c	WILC1000 driver OS abstraction layer.
wdrv_wilc1000_scan_helper.c	WILC1000 driver scan helper functions.
wdrext_wilc1000.c	WILC1000 driver extended functions.
wilc1000_task.c	WILC1000 driver task handler.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	The WILC1000 Wi-Fi Driver controller has no optional files.

Module Dependencies

The WILC1000 Wi-Fi Driver Library depends on the following modules:

- SPI Driver Library
- NVM Driver Library
- UART Driver Library
- USB Driver Library
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library Introduction
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Debug System Service Library
- Ports System Service Library
- FreeRTOS Library Help
- Crypto Library
- Peripheral Libraries
- Networking Presentation Layer Help
- TCP/IP Stack Library Help
- Command Processor System Service Library
- DMA System Service Library
- Random Number Generator System Service Library
- Common System Service Library
- TCP/IP Ethernet MAC Driver Library

Console Commands

This section describes the console commands available for the WILC1000 Wi-Fi Driver.

Description

Both the Web Server and the EasyConfig demonstrations support the followings commands, which enable control over the Wi-Fi settings.

Command: deleteconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.

[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example:
	`iwconfig security open`
	`iwconfig security wep40 <key>`
	`iwconfig security wep104 <key>`
	`iwconfig security wpa <key>`
	`iwconfig security wpa2 <key>`
	`iwconfig security pin <pin>`
	`iwconfig security pbc`

[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the MRF24WN module.

Command: ota

Parameters	Description
[http://ip-address/] [filename.bin]	Upgrade the WILC1000 firmware over-the-air. For example: http://192.168.0.4/winc1500_ota.bin

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Library Interface

This section describes the Application Programming Interface (API) functions of the WILC1000 Wi-Fi Driver.
Refer to each section for a detailed description.

a) Wi-Fi Initialization Functions

b) Wi-Fi Status Functions

c) Wi-Fi External Functions

d) Other Functions

e) Data Types and Constants

Files

Files

Name	Description
wdrv_wilc1000_api.h	WILC1000 wireless driver APIs.
wdrv_wilc1000_stub.h	WILC1000 wireless driver stub APIs.

Description

This section lists the source and header files used by the MRF24WN Wi-Fi Driver Library.

wdrv_wilc1000_api.h

WILC1000 wireless driver APIs.

Functions

	Name	Description
	WDRV_EXT_CmdScanOptionSet	Sets scan options. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the WILC1000 Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_ScanDoneSet	Indicates when a scan has completed. **Implementation:** Dynamic
	WDRV_EXT_ScanIsInProgress	Check whether host scan is now in progress or not. **Implementation:** Dynamic

Description

WILC1000 wireless driver APIs.

File Name

wdrv_wilc1000_api.h

Company

Microchip Technology Inc.

wdrv_wilc1000_stub.h

WILC1000 wireless driver stub APIs.

Description

WILC1000 wireless driver stub APIs.

File Name

wdrv_wilc1000_stub.h

Company

Microchip Technology Inc.

WINC1500 Wi-Fi Driver Ethernet Mode Library

This topic describes the WINC1500 Wi-Fi Driver Library.

Introduction

This library provides a low-level abstraction of the WINC1500 Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Note: The WINC1500 Wi-Fi Driver is compatible with the WINC1500 PICtail/PICtail Plus Daughter board with WINC1500 firmware version 19.5.2 and later in "Ethernet mode". The driver will also work with WINC1500 firmware version 19.4.4 with limited backward compatibility.

Description

The Wi-Fi software library, in conjunction with the WINC1500 module, allows an application to:

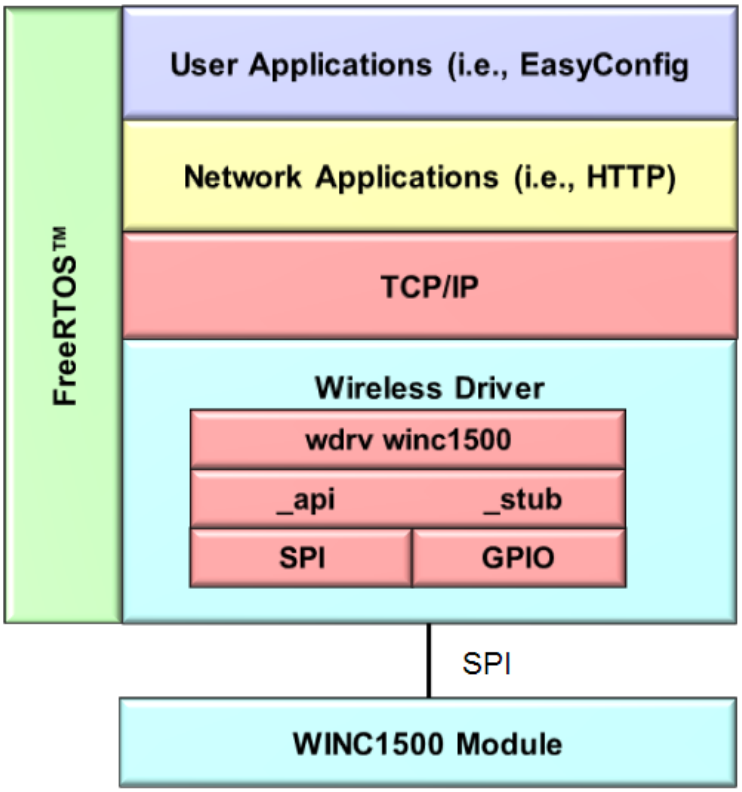
- Join an existing 802.11 Wi-Fi network
- Create a 802.11 Wi-Fi network

The following application services are provided by the Wi-Fi library:

- Configure a Wi-Fi connection (SSID, security mode, and so on)
- Join an existing network or create a "Soft-AP" Wi-Fi network
- Scan for other Wi-Fi devices in the area
- Receive Wi-Fi network status
- Wi-Fi power control

The MAC_layer services are not directly accessible to the application. This portion of the code resides under the TCP/IP Stack MAC module software layers and is used by stack services to transmit and receive data over a Wi-Fi network. The following diagram shows the interaction of the primary software blocks in a Wi-Fi application.

Wi-Fi Software Block Diagram



The following table provides information that includes network mode and security mode support by the WINC1500 Wi-Fi Driver.

WINC1500 Network Connection Matrix	Network Mode	
Security Mode	Infrastructure	SoftAP
Open	YES	YES
WEP40 (64-bit)	YES	YES
WEP104 (128-bit)	YES	YES
WPA-AUTO PSK (TKIP/AES)	YES	NA
WPA-AUTO PSK (AES)	YES	YES
WPS Push Button	YES	NA
WPS PIN	YES	NA

Using the Library

This topic describes the basic architecture of the WINC1500 Wi-Fi Driver Library and provides information and examples on its use.

Description

Interface Header Files: `wdrv_winc1500_api.h` and `wdrv_winc1500_stub.h`

The interface to the WINC1500 Wi-Fi Driver Library is defined in the `wdrv_winc1500_api.h` and `wdrv_winc1500_stub.h` header files.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the WINC1500 Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The WINC1500 Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing
- Wi-Fi status
- Wi-FI console commands

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

TheLibrary Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the Wi-Fi module.

Library Interface Section	Description
Wi-Fi Initialization Functions	This section provides functions that initialize the Wi-Fi library and allow its API to be used.
Wi-Fi Status Functions	This section provides functions that retrieve the Wi-Fi connection status.
Wi-Fi External Functions	This section provides public functions accessible to TCP/IP applications.
Other Functions	This section provides additional miscellaneous functions for configuring the Wi-Fi connection.
Data Types and Constants	This section provides data types and macros.

How the Library Works

This section describes how the WINC1500 Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, it should be configured (compile time configuration).

There are a few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the WINC1500 Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

Configuring the Library

This section describes how to configure the WINC1500 Wi-Fi driver.

Description

The configuration of the WINC1500 Wi-Fi Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the Wi-Fi Driver. Based on the selections made, the WINC1500 Wi-Fi Driver may support the selected features. These configuration settings will apply to all instances of the WINC1500 Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Sample Functionality

The following code provides an example of Wi-Fi Driver configuration. (refer to `system.config.h`)

```
/*** SPI Driver Configuration ***/
#define DRV_SPI_NUMBER_OF_MODULES        4

/*** Driver Compilation and static configuration options. ***/
/*** Select SPI compilation units.***/
#define DRV_SPI_POLLED           0
#define DRV_SPI_ISR              1
#define DRV_SPI_MASTER           1
#define DRV_SPI_SLAVE            0
#define DRV_SPI_RM               1
#define DRV_SPI_EBM              0
#define DRV_SPI_8BIT             1
#define DRV_SPI_16BIT            0
#define DRV_SPI_32BIT            0
#define DRV_SPI_DMA              1

/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER         1
#define DRV_SPI_CLIENTS_NUMBER           1
#define DRV_SPI_ELEMENTS_PER_QUEUE       10

/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE            512
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE     512

/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0                 SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0              DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0               DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0         false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0      DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0             SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0              CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0              2000000
#define DRV_SPI_BUFFER_TYPE_IDX0            DRV_SPI_BUFFER_TYPE_STANDARD
#define DRV_SPI_CLOCK_MODE_IDX0             DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0            SPI_INPUT_SAMPLING_PHASE_AT_END
#define DRV_SPI_TRANSMIT_DUMMY_BYTE_VALUE_IDX0      0x00

#define DRV_SPI_TX_INT_SOURCE_IDX0          INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0          INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0             INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0           INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0       INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0             10
#define DRV_SPI_RESERVED_JOB_IDX0           1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0         DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0       16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0         DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0       16
```

```
/*** Timer Driver Configuration ***/
#define DRV_TMR_INTERRUPT_MODE            true
#define DRV_TMR_INSTANCES_NUMBER          1
#define DRV_TMR_CLIENTS_NUMBER            1

/*** Timer Driver 0 Configuration ***/
#define DRV_TMR_PERIPHERAL_ID_IDX0          TMR_ID_2
#define DRV_TMR_INTERRUPT_SOURCE_IDX0       INT_SOURCE_TIMER_2
#define DRV_TMR_INTERRUPT_VECTOR_IDX0       INT_VECTOR_T2
#define DRV_TMR_ISR_VECTOR_IDX0             _TIMER_2_VECTOR
#define DRV_TMR_INTERRUPT_PRIORITY_IDX0     INT_PRIORITY_LEVEL4
#define DRV_TMR_INTERRUPT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_TMR_CLOCK_SOURCE_IDX0           DRV_TMR_CLKSOURCE_INTERNAL
#define DRV_TMR_PRESCALE_IDX0               TMR_PRESCALE_VALUE_256
#define DRV_TMR_OPERATION_MODE_IDX0         DRV_TMR_OPERATION_MODE_16_BIT
#define DRV_TMR_ASYNC_WRITE_ENABLE_IDX0     false
#define DRV_TMR_POWER_STATE_IDX0            SYS_MODULE_POWER_RUN_FULL

 /*** Wi-Fi Driver Configuration ***/
#define WINC1500_INT_SOURCE INT_SOURCE_CHANGE_NOTICE
#define WINC1500_INT_VECTOR INT_VECTOR_CN

#define WDRV_SPI_INDEX 0
#define WDRV_SPI_INSTANCE sysObj.spiObjectIdx0

#define WDRV_USE_SPI_DMA

#define WDRV_NVM_SPACE_ENABLE
#define WDRV_NVM_SPACE_ADDR (48 * 1024)

#define WDRV_BOARD_TYPE WDRV_BD_TYPE_MX_ESK

#define WDRV_EXT_RTOS_TASK_SIZE 2048u
#define WDRV_EXT_RTOS_TASK_PRIORITY 2u

// I/O mappings for general control pins, including CHIP_EN, IRQN, RESET_N and SPI_SSN.
#define WDRV_CHIP_EN_PORT_CHANNEL   PORT_CHANNEL_F
#define WDRV_CHIP_EN_BIT_POS        1

#define WDRV_IRQN_PORT_CHANNEL      PORT_CHANNEL_G
#define WDRV_IRQN_BIT_POS           7

#define WDRV_RESET_N_PORT_CHANNEL   PORT_CHANNEL_F
#define WDRV_RESET_N_BIT_POS        0

#define WDRV_SPI_SSN_PORT_CHANNEL   PORT_CHANNEL_B
#define WDRV_SPI_SSN_BIT_POS        2

#define WINC1500_ON_PIC32MX_ESK

// On PIC32MX ESK, when CN9 (Pin G7) is used as external interrupt,
// it is sometimes better to use another GPIO (Pin E0) to read CN9's value.
// In this case, a jumper wire is needed to connect Pin E0 and Pin G7.
//#define WDRV_VERIFY_IRQN_BY_ANOTHER_GPIO
#if defined(WDRV_VERIFY_IRQN_BY_ANOTHER_GPIO)
// Use Pin E0. Please also make sure that Pin E0 and Pin G7 are connected (by a jumper wire).
#define WDRV_IRQN_PORT_CHANNEL_READ     PORT_CHANNEL_E
#define WDRV_IRQN_BIT_POS_READ          0
#else
// Still directly read Pin G7's value.
#define WDRV_IRQN_PORT_CHANNEL_READ     PORT_CHANNEL_G
#define WDRV_IRQN_BIT_POS_READ          7
#endif

#define WDRV_DEFAULT_NETWORK_TYPE WDRV_NETWORK_TYPE_INFRASTRUCTURE
#define WDRV_DEFAULT_CHANNEL 6
#define WDRV_DEFAULT_SSID "MicrochipDemoApp"
```

```
#define WDRV_DEFAULT_SECURITY_MODE WDRV_SECURITY_OPEN
#define WDRV_DEFAULT_WEP_KEYS_40 "5AFB6C8E77" // default WEP40 key
#define WDRV_DEFAULT_WEP_KEYS_104 "90E96780C739409DA50034FCAA" // default WEP104 key
#define WDRV_DEFAULT_PSK_PHRASE "Microchip 802.11 Secret PSK Password" // default WPA-PSK
```

Building the Library

This section lists the files that are available in the WINC1500 Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/winc1500`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
wdrv_winc1500_stub.h	Contains Stub function prototypes for interfacing to the WINC1500 Wi-Fi Driver.
wdrv_winc1500_api.h	Contains API function prototypes for interfacing to the WINC1500 Wi-Fi Driver.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
wdrv_winc1500_console.c	Console module for the WINC1500 wireless driver.
wdrv_winc1500_fw_update.c	WINC1500 firmware update support.
wdrv_winc1500_eint.c	External interrupt handler for the WINC1500 wireless driver.
wdrv_winc1500_timer.c	Timer functions for the WINC1500 wireless driver.
wdrv_winc1500_gpio.c	WINC1500 GPIO support for SPI communication.
wdrv_winc1500_spi.c	WINC1500 support for SPI communication.
wdrv_winc1500_cli.c	WINC1500 driver CLI implementation.
wdrv_winc1500_config_data.c	Stores and retrieves Wi-Fi configuration to/from non-volatile memory (NVM).
wdrv_winc1500_connmgr.c	WINC1500 driver connection manager.
wdrv_winc1500_events.c	WINC1500 driver MAC events.
wdrv_winc1500_iwpriv.c	WINC1500 driver connection process functions.
wdrv_winc1500_main.c	WINC1500 driver Microchip TCP/IP Stack PIC32 MAC support.
wdrv_winc1500_osal.c	WINC1500 driver OS abstraction layer.
wdrv_winc1500_scan_helper.c	WINC1500 driver scan helper functions.
wdrext_winc1500.c	WINC1500 driver extended functions.
winc1500_task.c	WINC1500 driver task handler.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	The WINC1500 Wi-Fi Driver controller has no optional files.

Module Dependencies

The WINC1500 Wi-Fi Driver Library depends on the following modules:

- SPI Driver Library
- NVM Driver Library

- UART Driver Library
- USB Driver Library
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library Introduction
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Debug System Service Library
- Ports System Service Library
- FreeRTOS Library Help
- Crypto Library
- Peripheral Libraries
- Networking Presentation Layer Help
- TCP/IP Stack Library Help
- Command Processor System Service Library
- DMA System Service Library
- Random Number Generator System Service Library
- Common System Service Library
- TCP/IP Ethernet MAC Driver Library

Console Commands

This section describes the console commands available for the WINC1500 Wi-Fi Driver.

Description

Both the Web Server and the EasyConfig demonstrations support the followings commands, which enable control over the Wi-Fi settings.

Command: deleteconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.

[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example:
	`iwconfig security open`
	`iwconfig security wep40 <key>`
	`iwconfig security wep104 <key>`
	`iwconfig security wpa <key>`
	`iwconfig security wpa2 <key>`
	`iwconfig security pin <pin>`
	`iwconfig security pbc`

[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the MRF24WN module.

Command: ota

Parameters	Description
[http://ip-address/] [filename.bin]	Upgrade the WINC1500 firmware over-the-air. For example: http://192.168.0.4/winc1500_ota.bin

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Library Interface

a) Wi-Fi Initialization Functions

	Name	Description
	WDRV_CLI_Init	Initializes the console CLI interface. **Implementation:** Dynamic
	WDRV_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_Deinit	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_Init	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_DeInit	Deinitializes the GPIO objects for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_Deinitialize	Deinitializes the WINC1500 Wi-Fi driver. **Implementation:** Dynamic
	WDRV_WINC1500_ISR	Wi-Fi driver (WINC1500-specific) interrupt service routine. **Implementation:** Dynamic

b) Wi-Fi Status Functions

	Name	Description
	WDRV_EXT_CmdFWVersionGet	Retrieves FW version information. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_CmdMacAddressGet	Retrieves the WINC1500 MAC address. **Implementation:** Dynamic
	WDRV_EXT_CmdScanGet	Reads the number of scan results from the WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDGet	Gets the SSID. **Implementation:** Dynamic

c) Wi-Fi External Functions

	Name	Description
	WDRV_EXT_CmdPowerSavePut	Puts the module in power save mode. **Implementation:** Dynamic
	WDRV_EXT_HWInterruptHandler	Wi-Fi driver (WINC1500-specific) interrupt service routine. **Implementation:** Dynamic

	WDRV_EXT_CmdScanOptionSet	Sets scan options. **Implementation:** Dynamic
	WDRV_EXT_ModuleUpDown	Enables or disables WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_MulticastFilterSet	Sets a multicast address filter. **Implementation:** Dynamic
	WDRV_EXT_CmdConnect	Directs the WINC1500 to connect to a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdDisconnect	Directs the WINC1500 to disconnect from a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeAPSet	Sets the Wi-Fi network type to SoftAP. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeBSSSet	Sets the Wi-Fi network type to Infrastructure. **Implementation:** Dynamic
	WDRV_EXT_CmdScanStart	Directs the WINC1500 module to start a scan. **Implementation:** Dynamic
	WDRV_EXT_CmdSecNoneSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	WDRV_EXT_CmdSecWEPSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPASet	Sets Wi-Fi security to use WPA/WPA2. **Implementation:** Dynamic
	WDRV_EXT_DataSend	Sends data packets to WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_ScanDoneSet	Indicates when a scan has completed. **Implementation:** Dynamic
	WDRV_EXT_CmdChannelSet	Sets the channel on which to operate. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdFWUpdate	Directs the module to start firmware download and upgrade. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	WDRV_EXT_CmdTxPowerSet	Sets the Tx Power at 3 levels, high, medium and low. **Implementation:** Dynamic
	WDRV_EXT_CmdConnectContextBssidGet	Gets the BSSID **Implementation:** Dynamic
	WDRV_EXT_CmdScanOptionsSet	Sets scan options. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_ScanIsInProgress	Check whether host scan is now in progress or not. **Implementation:** Dynamic

d) Other Functions

	Name	Description
	WDRV_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic
	WDRV_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the WILC1000 Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_RssiRead	Requests RSSI for the connected AP. **Implementation:** Dynamic
	WDRV_EXT_WPSResultsRead	Reads the WPS process results back from the WINC1500 module and updates the configuration data. **Implementation:** Dynamic

	WDRV_STUB_Assert	Dumps out an error message on serial console and resets itself when the driver asserts. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ChipDisable	Disables the WINC1500 chip. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ChipEnable	Enables the WINC1500 chip. **Implementation:** Dynamic
	WDRV_STUB_GPIO_DeInitialize	Deinitializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_GPIO_Initialize	Initializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ModuleReset	Resets the WINC1500 module. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ModuleUnreset	Unresets the WINC1500 module. **Implementation:** Dynamic
	WDRV_STUB_HardDelay	Waits spinning for the delay milliseconds. **Implementation:** Dynamic
	WDRV_STUB_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic
	WDRV_STUB_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_STUB_SPI_Deinitialize	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic
	WDRV_STUB_SPI_Initialize	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	_WDRV_WINC1500_API_H	This is macro _WDRV_WINC1500_API_H.
	WDRV_STUB_Print	This is macro WDRV_STUB_Print.

Description

This section describes the Application Programming Interface (API) functions of the WINC1500 Wi-Fi Driver.

Refer to each section for a detailed description.

a) Wi-Fi Initialization Functions

WDRV_CLI_Init Function

Initializes the console CLI interface.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
bool WDRV_CLI_Init();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function initializes the console CLI interface.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

bool WDRV_CLI_Init(void)

WDRV_INTR_Deinit Function

Deinitializes interrupts for Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_Deinit();
```

Returns

None.

Description

This function deinitializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_INTR_Deinit(void)

WDRV_INTR_Init Function

Initializes interrupts for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_Init();
```

Returns

None.

Description

This function initializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_INTR_Init(void)

WDRV_SPI_Deinit Function

Deinitializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Deinit();
```

Returns

None.

Description

This function deinitializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_SPI_Deinit(void)

WDRV_SPI_Init Function

Initializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Init();
```

Returns

None.

Description

This function initializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_SPI_Init(void)

WDRV_GPIO_DeInit Function

Deinitializes the GPIO objects for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_DeInit();
```

Returns

None.

Description

This function deinitializes the GPIO objects for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_DeInit(void)

WDRV_EXT_Deinitialize Function

Deinitializes the WINC1500 Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_Deinitialize();
```

Returns

None.

Description

This function deinitializes the WINC1500 driver.

Remarks

None

Preconditions

None.

Function

void WDRV_EXT_Deinitialize(void)

WDRV_WINC1500_ISR Function

Wi-Fi driver (WINC1500-specific) interrupt service routine.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_WINC1500_ISR();
```

Returns

None.

Description

This function is the Wi-Fi driver (WINC1500-specific) interrupt service routine.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_WINC1500_ISR(void)

b) Wi-Fi Status Functions

WDRV_EXT_CmdFWVersionGet Function

Retrieves FW version information.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdFWVersionGet(uint32_t * major, uint32_t * minor, uint32_t * patch);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function retrieves the module FW version information.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
major	pointer where the major number will be written
minor	pointer where the minor number will be written
patch	pointer where the patch number will be written

Function

uint32_t WDRV_EXT_CmdFWVersionGet(uint32_t *major, uint32_t *minor, uint32_t *patch);

WDRV_EXT_ScanResultGet Function

Reads the selected scan results back from the WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_ScanResultGet(uint8_t listIndex, WDRV_SCAN_RESULT * p_scanResult);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

After a scan has completed this function is used to read one scan result at a time from the WINC1500 module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
listIndex	index (0 based list) of the scan entry to retrieve
p_scanResult	pointer to where scan result is written

Function

void WDRV_EXT_ScanResultGet(uint8_t listIndex, WDRV_SCAN_RESULT *p_scanResult)

WDRV_EXT_CmdMacAddressGet Function

Retrieves the WINC1500 MAC address.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdMacAddressGet(uint8_t * MacAddr);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function retrieves the WINC1500 MAC address.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
MacAddr	Pointer where MAC address will be written (must point to a 6 bytes buffer)

Function

uint32_t WDRV_EXT_CmdMacAddressGet(uint8_t *MacAddr)

WDRV_EXT_CmdScanGet Function

Reads the number of scan results from the WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdScanGet(uint16_t * numOfResults);
```

Returns

None.

Description

This function reads the number of scan results from the WINC1500 module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
numOfResults	pointer where the number of scan results will be written

Function

void WDRV_EXT_CmdScanGet(uint16_t *numOfResults)

WDRV_EXT_CmdSSIDGet Function

Gets the SSID.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdSSIDGet(uint8_t * ssid, uint8_t * length);
```

Returns

None.

Description

This function returns the SSID and SSID Length.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to buffer where SSID will be written
length	number of bytes in SSID

Function

void WDRV_EXT_CmdSSIDGet(uint8_t *ssid, uint8_t *length)

c) Wi-Fi External Functions

WDRV_EXT_CmdPowerSavePut Function

Puts the module in power save mode.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdPowerSavePut(bool enable, uint8_t mode, uint16_t listenInterval);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function places the module in power save mode.

Remarks

This works only with Infrastructure mode. Do not call this in other modes.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enable	true will put the module in power save mode.
mode	0 : manual mode - not synchronized to AP beacon ; 1. deep automatic mode - ieee802.11 power save mode.
listenInterval	STA wakes up per this beacon interval.

Function

uint32_t WDRV_EXT_CmdPowerSavePut(bool enable, uint8_t mode, uint16_t listenInterval)

WDRV_EXT_HWInterruptHandler Function

Wi-Fi driver (WINC1500-specific) interrupt service routine.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_HWInterruptHandler();
```

Returns

None.

Description

This function is the Wi-Fi driver (WINC1500-specific) interrupt service routine.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_HWInterruptHandler(void)

WDRV_EXT_CmdScanOptionSet Function

Sets scan options.

Implementation: Dynamic

File

wdrv_wilc1000_api.h

C

```
uint32_t WDRV_EXT_CmdScanOptionSet(uint8_t numOfSlots, uint8_t slotTime, uint8_t probesPerSlot, uint8_t
rssiThreshold);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function sets scan options.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
numOfSlots	The min number of slots is 2 for every channel, every slot the module will send Probe Request on air, and wait/listen for PROBE RESP/BEACONS for the slotTime.
slotTime	The time that the module will wait on every channel listening to the frames on air.
probesPerSlot	Number of probe requests to be sent per channel scan slot.
rssiThreshold	The RSSI threshold of the AP which will be connected to directly.

Function

uint32_t WDRV_EXT_CmdScanOptionSet(uint8_t numOfSlots, uint8_t slotTime, uint8_t probesPerSlot, uint8_t rssiThreshold);

WDRV_EXT_ModuleUpDown Function

Enables or disables WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_ModuleUpDown(uint32_t up);
```

Returns

None.

Description

This function enables or disables WINC1500 module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
up	1: enable; 0: disable.

Function

void WDRV_EXT_ModuleUpDown(uint32_t up)

WDRV_EXT_MulticastFilterSet Function

Sets a multicast address filter.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_MulticastFilterSet(uint8_t * addr);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function allows the application to configure up to 8 Multicast address filters on the WINC1500 module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
addr	the pointer of the multicast mac address.

Function

uint32_t WDRV_EXT_MulticastFilterSet(uint8_t *addr)

WDRV_EXT_CmdConnect Function

Directs the WINC1500 to connect to a Wi-Fi network.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdConnect();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function causes the WINC1500 to connect to a Wi-Fi network. Upon connection, or a failure to connect, an event will be generated.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and relevant connection parameters must have been set.

Function

uint32_t WDRV_EXT_CmdConnect(void)

WDRV_EXT_CmdDisconnect Function

Directs the WINC1500 to disconnect from a Wi-Fi network.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdDisconnect();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function causes the WINC1500 to disconnect from a Wi-Fi network.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and a connection must be in progress.

Function

uint32_t WDRV_EXT_CmdDisconnect(void)

WDRV_EXT_CmdNetModeAPSet Function

Sets the Wi-Fi network type to SoftAP.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdNetModeAPSet();
```

Returns

None.

Description

This function sets the Wi-Fi network type to SoftAP.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_CmdNetModeAPSet(void)

WDRV_EXT_CmdNetModeBSSSet Function

Sets the Wi-Fi network type to Infrastructure.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdNetModeBSSSet();
```

Returns

None.

Description

This function sets the Wi-Fi network type to Infrastructure.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_CmdNetModeBSSSet(void)

WDRV_EXT_CmdScanStart Function

Directs the WINC1500 module to start a scan.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdScanStart();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function directs the WINC1500 module to start a scan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdScanStart(void)

WDRV_EXT_CmdSecNoneSet Function

Sets Wi-Fi security to open (no security).

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdSecNoneSet();
```

Returns

None.

Description

This function sets the Wi-Fi security to open. One can only connect to an AP that is running in open mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Function

void WDRV_EXT_CmdSecNoneSet(void)

WDRV_EXT_CmdSecWEPSet Function

Sets Wi-Fi security to use WEP.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdSecWEPSet(uint8_t * key, uint16_t len);
```

Returns

None.

Description

This function sets the Wi-Fi security to WEP. One can only connect to an AP that is running the same WEP mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WEP key buffer
len	WEP key length

Function

void WDRV_EXT_CmdSecWEPSet(uint8_t *key, uint16_t len)

WDRV_EXT_CmdSecWPASet Function

Sets Wi-Fi security to use WPA/WPA2.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdSecWPASet(uint8_t * key, uint16_t len);
```

Returns

None.

Description

This function sets the Wi-Fi security to WPA/WPA2. One can only connect to an AP that is running the same WPA/WPA2 mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WPA key buffer
len	WPA key length

Function

void WDRV_EXT_CmdSecWPASet(uint8_t *key, uint16_t len)

WDRV_EXT_DataSend Function

Sends data packets to WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_DataSend(uint16_t segSize, uint8_t * p_segData);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sends data packets to the WINC1500 module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
seqSize	data size
p_seqData	pointer to the data buffer

Function

uint32_t WDRV_EXT_DataSend(uint16_t segSize, uint8_t *p_segData)

WDRV_EXT_ScanDoneSet Function

Indicates when a scan has completed.

Implementation: Dynamic

File

wdrv_wilc1000_api.h

C

```
void WDRV_EXT_ScanDoneSet();
```

Returns

None.

Description

This function indicates when a scan has completed.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_ScanDoneSet(void)

WDRV_EXT_CmdChannelSet Function

Sets the channel on which to operate.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdChannelSet(uint16_t channel);
```

Returns

None.

Description

This function sets the channel on which to operate.

Remarks

This works only with SoftAP mode. Do not call this in other modes.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
channel	target channel

Function

void WDRV_EXT_CmdChannelSet(uint16_t channel)

WDRV_EXT_CmdSSIDSet Function

Sets the SSID.

Implementation: Dynamic

File

wdrv_wilc1000_api.h

C

```
void WDRV_EXT_CmdSSIDSet(uint8_t * ssid, uint16_t len);
```

Returns

None.

Description

This function sets the SSID and SSID length.

Remarks

Do not include a string terminator in the SSID length. SSIDs are case-sensitive. SSID length must be less than or equal to 32.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to SSID buffer
len	number of bytes in SSID

Function

void WDRV_EXT_CmdSSIDSet(uint8_t *ssid, uint16_t len)

WDRV_EXT_CmdFWUpdate Function

Directs the module to start firmware download and upgrade.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdFWUpdate();
```

Returns

None.

Description

This function directs the module to start the firmware download and upgrade.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_CmdFWUpdate(void)

WDRV_EXT_CmdSecWpsSet Function

Sets Wi-Fi security to use WPS.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdSecWpsSet(bool pinMode, uint8_t * key, uint16_t keyLen);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WPS. One can only connect to an AP that supports WPS.

Remarks

None

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
pinMode	0: PBC mode; 1: PIN mode
key	pointer of the PIN buffer
keyLen	PIN length

Function

int32_t WDRV_EXT_CmdSecWpsSet(bool pinMode, uint8_t *key, uint16_t keyLen)

WDRV_EXT_CmdTxPowerSet Function

Sets the Tx Power at 3 levels, high, medium and low.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdTxPowerSet(uint32_t level);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function sets the module's Tx power.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
level	1 : high - 18 dBm PA gain , 2 : medium - 12 dBm PA gain, 3 : low - 6 dBm PA gain.

Function

uint32_t WDRV_EXT_CmdTxPowerSet(uint32_t level)

WDRV_EXT_CmdConnectContextBssidGet Function

Gets the BSSID

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdConnectContextBssidGet(uint8_t * bssId);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function gets the current AP's BSSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
bssId	pointer where the current AP's BSSID will be written

Function

uint32_t WDRV_EXT_CmdConnectContextBssidGet(uint8_t *bssId)

WDRV_EXT_CmdScanOptionsSet Function

Sets scan options.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_CmdScanOptionsSet(uint8_t numOfSlots, uint8_t slotTime, uint8_t probesPerSlot, uint8_t
rssiThreshold);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function sets scan options.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
numOfSlots	The min number of slots is 2 for every channel, every slot the module will send Probe Request on air, and wait/listen for PROBE RESP/BEACONS for the slotTime.
slotTime	The time that the module will wait on every channel listening to the frames on air.
probesPerSlot	Number of probe requests to be sent per channel scan slot.
rssiThreshold	The RSSI threshold of the AP which will be connected to directly.

Function

uint32_t WDRV_EXT_CmdScanOptionsSet(uint8_t numOfSlots, uint8_t slotTime, uint8_t probesPerSlot, uint8_t rssiThreshold);

WDRV_EXT_CmdSSIDSet Function

Sets the SSID.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_CmdSSIDSet(uint8_t * ssid, uint8_t len);
```

Returns

None.

Description

This function sets the SSID and SSID length.

Remarks

SSIDs are case-sensitive. SSID length must be less than or equal to 32.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to SSID buffer
len	number of bytes in SSID

Function

void WDRV_EXT_CmdSSIDSet(uint8_t *ssid, uint16_t len)

WDRV_EXT_ScanIsInProgress Function

Check whether host scan is now in progress or not.

Implementation: Dynamic

File

wdrv_wilc1000_api.h

C

```
bool WDRV_EXT_ScanIsInProgress();
```

Returns

- true - Host scan is in progress
- false - Host scan is not in progress

Description

Check whether host scan is now in progress or not.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_ScanIsInProgress(void)

d) Other Functions

WDRV_INTR_SourceDisable Function

Disables interrupts from the module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_SourceDisable();
```

Returns

None.

Description

This function disables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_INTR_SourceDisable(void)

WDRV_INTR_SourceEnable Function

Enables interrupts from the module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_SourceEnable();
```

Returns

None.

Description

This function enables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_INTR_SourceEnable(void)

WDRV_EXT_Initialize Function

Initializes the WILC1000 Wi-Fi driver.

Implementation: Dynamic

File

wdrv_wilc1000_api.h

C

```
void WDRV_EXT_Initialize(WDRV_HOOKS const *const ehooks, bool initWait);
```

Returns

None.

Description

This function initializes the WILC1000 Wi-Fi driver, making it ready for clients to use.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
ehooks	pointer to WDRV layer hooks
initWait	true will put WDRV in wait during initialization

Function

void WDRV_EXT_Initialize(WDRV_HOOKS const *const ehooks, bool initWait)

WDRV_EXT_RssiRead Function

Requests RSSI for the connected AP.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
uint32_t WDRV_EXT_RssiRead();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function requests RSSI for the connected AP.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_RssiRead(void)

WDRV_EXT_WPSResultsRead Function

Reads the WPS process results back from the WINC1500 module and updates the configuration data.

Implementation: Dynamic

File

wdrv_winc1500_api.h

C

```
void WDRV_EXT_WPSResultsRead(WDRV_CONFIG * config, uint32_t * status);
```

Returns

None.

Description

After the WPS process has completed, this function is used to read the WPS process results from the WINC1500 module and update the configuration data.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
config	pointer to where configuration data will be updated
status	pointer to where WPS process status will be written

Function

void WDRV_EXT_WPSResultsRead(WDRV_CONFIG *config, uint32_t *status)

WDRV_STUB_Assert Function

Dumps out an error message on serial console and resets itself when the driver asserts.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_Assert(int condition, const char * msg, const char * file, int line);
```

Returns

None.

Description

Dumps out an error message on serial console and resets itself when the driver asserts.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
condition	asserts if false
msg	error message
file	file name
line	line number where driver asserts.

Function

WDRV_STUB_Assert(int condition, const char *msg, const char *file, int line)

WDRV_STUB_GPIO_ChipDisable Function

Disables the WINC1500 chip.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_ChipDisable();
```

Returns

None.

Description

This function disables the WINC1500 chip.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_GPIO_ChipDisable(void)

WDRV_STUB_GPIO_ChipEnable Function

Enables the WINC1500 chip.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_ChipEnable();
```

Returns

None.

Description

This function enables the WINC1500 chip.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_GPIO_ChipEnable(void)

WDRV_STUB_GPIO_DeInitialize Function

Deinitializes the GPIO object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_DeInitialize();
```

Returns

None.

Description

This function deinitializes the GPIO object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_GPIO_DeInitialize(void)

WDRV_STUB_GPIO_Initialize Function

Initializes the GPIO object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_Initialize();
```

Returns

None.

Description

This function initializes the GPIO object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_GPIO_Initialize(void)

WDRV_STUB_GPIO_ModuleReset Function

Resets the WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_ModuleReset();
```

Returns

None.

Description

This function resets the WINC1500 module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_GPIO_ModuleReset(void)

WDRV_STUB_GPIO_ModuleUnreset Function

Unresets the WINC1500 module.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_GPIO_ModuleUnreset();
```

Returns

None.

Description

This function unresets the WINC1500 module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
board	Microchip development kit type (i.e., PIC32MZ EC Starter Kit, PIC32 Ethernet Starter Kit, etc.)

Function

void WDRV_STUB_GPIO_ModuleUnreset(void)

WDRV_STUB_HardDelay Function

Waits spinning for the delay milliseconds.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_HardDelay(uint16_t delay);
```

Returns

None.

Description

This function has driver wait spinining for the delay milliseconds.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
board	duration to spin.

Function

WDRV_STUB_HardDelay(uint16_t delay)

WDRV_STUB_INTR_Deinit Function

Deinitializes interrupts for Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_INTR_Deinit();
```

Returns

None.

Description

This function deinitializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_STUB_INTR_Deinit(void)

WDRV_STUB_INTR_Init Function

Initializes interrupts for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_INTR_Init(void (*isr)(void));
```

Returns

None.

Description

This function initializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
isr	function pointer to the interrupt service handler.

Function

void WDRV_STUB_INTR_Init(void (*isr)(void))

WDRV_STUB_INTR_SourceDisable Function

Disables interrupts from the module.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_INTR_SourceDisable();
```

Returns

None.

Description

This function disables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_STUB_INTR_SourceDisable(void)

WDRV_STUB_INTR_SourceEnable Function

Enables interrupts from the module.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_INTR_SourceEnable();
```

Returns

None.

Description

This function enables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_STUB_INTR_SourceEnable(void)

WDRV_STUB_SPI_Deinitialize Function

Deinitializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_SPI_Deinitialize();
```

Returns

None.

Description

This function deinitializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

None.

Function

void WDRV_STUB_SPI_Deinitialize(void)

WDRV_STUB_SPI_In Function

Receives data from the module through the SPI bus.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
bool WDRV_STUB_SPI_In(unsigned char *const buf, uint32_t size);
```

Returns

None.

Description

This function receives data from the module through the SPI bus.

Remarks

None.

Preconditions

SPI driver should be initialized.

Parameters

Parameters	Description
buf	buffer pointer of input data
size	the input data size

Function

bool WDRV_STUB_SPI_In(unsigned char *const buf, uint32_t size)

WDRV_STUB_SPI_Initialize Function

Initializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
void WDRV_STUB_SPI_Initialize();
```

Returns

None.

Description

This function initializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

None.

Function

void WDRV_STUB_SPI_Initialize(void)

WDRV_STUB_SPI_Out Function

Sends data out to the module through the SPI bus.

Implementation: Dynamic

File

wdrv_winc1500_stub.h

C

```
bool WDRV_STUB_SPI_Out(unsigned char *const buf, uint32_t size);
```

Returns

True - Indicates success False - Indicates failure

Description

This function sends data out to the module through the SPI bus.

Remarks

None.

Preconditions

SPI driver should be initialized.

Parameters

Parameters	Description
buf	buffer pointer of output data
size	the output data size

Function

bool WDRV_STUB_SPI_Out(unsigned char const *buf, uint32_t size)

e) Data Types and Constants

_WDRV_WINC1500_API_H Macro

File

wdrv_winc1500_api.h

C

```
#define _WDRV_WINC1500_API_H
```

Description

This is macro _WDRV_WINC1500_API_H.

WDRV_STUB_Print Macro

File

wdrv_winc1500_stub.h

C

```
#define WDRV_STUB_Print(x) SYS_CONSOLE_PRINT x
```

Description

This is macro WDRV_STUB_Print.

Files

Files

Name	Description
wdrv_winc1500_api.h	WINC1500 wireless driver APIs.
wdrv_winc1500_stub.h	WINC1500 wireless driver stub APIs.

Description

This section lists the source and header files used by the MRF24WN Wi-Fi Driver Library.

wdrv_winc1500_api.h

WINC1500 wireless driver APIs.

Functions

	Name	Description
	WDRV_CLI_Init	Initializes the console CLI interface. **Implementation:** Dynamic
	WDRV_EXT_CmdChannelSet	Sets the channel on which to operate. **Implementation:** Dynamic
	WDRV_EXT_CmdConnect	Directs the WINC1500 to connect to a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdConnectContextBssidGet	Gets the BSSID **Implementation:** Dynamic
	WDRV_EXT_CmdDisconnect	Directs the WINC1500 to disconnect from a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdFWUpdate	Directs the module to start firmware download and upgrade. **Implementation:** Dynamic
	WDRV_EXT_CmdFWVersionGet	Retrieves FW version information. **Implementation:** Dynamic
	WDRV_EXT_CmdMacAddressGet	Retrieves the WINC1500 MAC address. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeAPSet	Sets the Wi-Fi network type to SoftAP. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeBSSSet	Sets the Wi-Fi network type to Infrastructure. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSavePut	Puts the module in power save mode. **Implementation:** Dynamic
	WDRV_EXT_CmdScanGet	Reads the number of scan results from the WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_CmdScanOptionsSet	Sets scan options. **Implementation:** Dynamic
	WDRV_EXT_CmdScanStart	Directs the WINC1500 module to start a scan. **Implementation:** Dynamic
	WDRV_EXT_CmdSecNoneSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	WDRV_EXT_CmdSecWEPSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPASet	Sets Wi-Fi security to use WPA/WPA2. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDGet	Gets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdTxPowerSet	Sets the Tx Power at 3 levels, high, medium and low. **Implementation:** Dynamic
	WDRV_EXT_DataSend	Sends data packets to WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_Deinitialize	Deinitializes the WINC1500 Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_HWInterruptHandler	Wi-Fi driver (WINC1500-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the WINC1500 Wi-Fi driver. **Implementation:** Dynamic

	WDRV_EXT_ModuleUpDown	Enables or disables WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_MulticastFilterSet	Sets a multicast address filter. **Implementation:** Dynamic
	WDRV_EXT_RssiRead	Requests RSSI for the connected AP. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the WINC1500 module. **Implementation:** Dynamic
	WDRV_EXT_WPSResultsRead	Reads the WPS process results back from the WINC1500 module and updates the configuration data. **Implementation:** Dynamic
	WDRV_WINC1500_ISR	Wi-Fi driver (WINC1500-specific) interrupt service routine. **Implementation:** Dynamic

Macros

	Name	Description
	_WDRV_WINC1500_API_H	This is macro _WDRV_WINC1500_API_H.

Description

WINC1500 wireless driver APIs.

File Name

wdrv_winc1500_api.h

Company

Microchip Technology Inc.

wdrv_winc1500_stub.h

WINC1500 wireless driver stub APIs.

Functions

	Name	Description
	WDRV_STUB_Assert	Dumps out an error message on serial console and resets itself when the driver asserts. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ChipDisable	Disables the WINC1500 chip. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ChipEnable	Enables the WINC1500 chip. **Implementation:** Dynamic
	WDRV_STUB_GPIO_DeInitialize	Deinitializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_GPIO_Initialize	Initializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ModuleReset	Resets the WINC1500 module. **Implementation:** Dynamic
	WDRV_STUB_GPIO_ModuleUnreset	Unresets the WINC1500 module. **Implementation:** Dynamic
	WDRV_STUB_HardDelay	Waits spinning for the delay milliseconds. **Implementation:** Dynamic
	WDRV_STUB_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic
	WDRV_STUB_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_STUB_SPI_Deinitialize	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic

	WDRV_STUB_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic
	WDRV_STUB_SPI_Initialize	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_STUB_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic

Macros

	Name	Description
	WDRV_STUB_Print	This is macro WDRV_STUB_Print.

Description

WINC1500 wireless driver stub APIs.

File Name

wdrv_winc1500_stub.h

Company

Microchip Technology Inc.

WINC1500 Socket Mode Driver Library

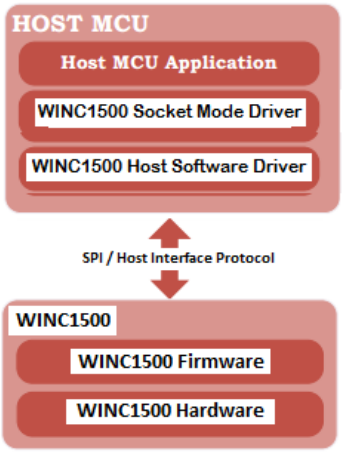
This section provides documentation for the WINC1500 Socket Mode Driver Library.

Introduction

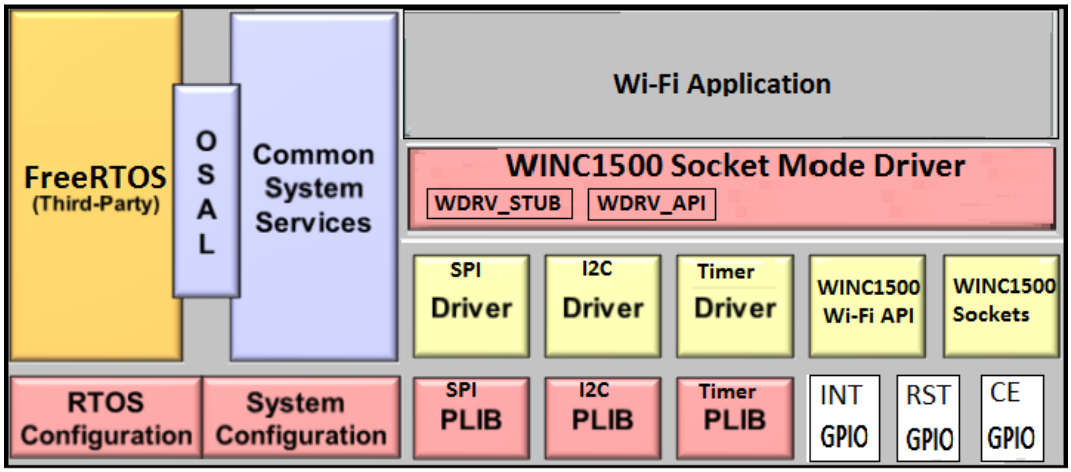
This library provides a low-level abstraction of the WINC1500 Socket Mode Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

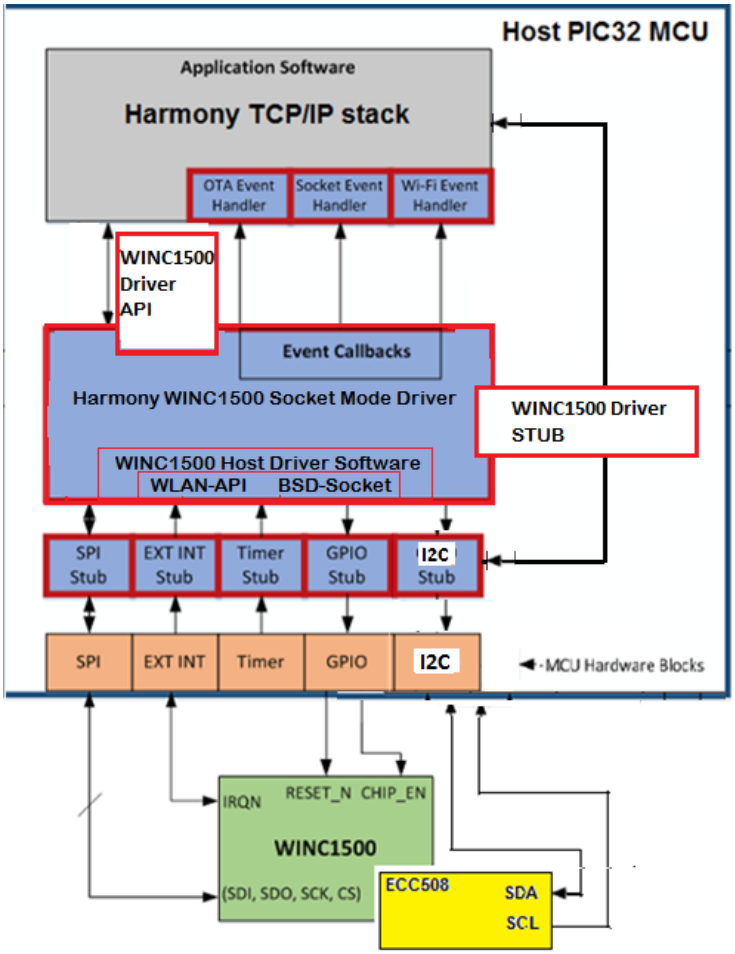
The WINC1500 Socket Mode Driver library is a C library provides the host MCU application with APIs for WLAN and socket operations, off-loading the host MCU TCP/IP networking and transport layer operations to the WINC1500 module firmware.



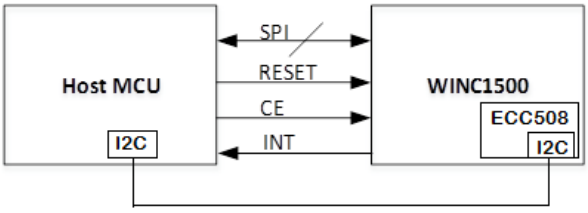
The MPLAB Harmony Integrated Software Framework blocks for the WINC1500 Wi-Fi Application are shown in the following diagram.



The following diagram shows the partitioning of the MPLAB Harmony WINC1500 Socket Mode Driver software on a MCU. Further discussions reference this diagram.



In the previous diagram, the WINC1500 module requires only a SPI interface, a timer, 2 GPIOs for CE and INT, and an interrupt line to connect to the host PIC32 MCU, as shown in the following figure. The figure also shows the I2C interface between the ECC508 device and the Host MCU; however, this feature is not available in the current release and will be available in a future release MPLAB Harmony.



Using the Library

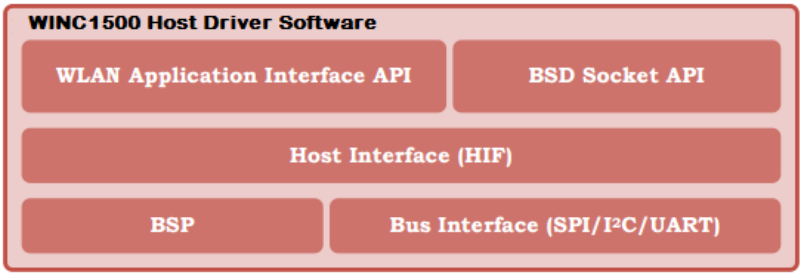
This topic describes the basic architecture of the WINC1500 Wi-Fi Driver Library and provides information and examples on its use.

Description

The WINC1500 Socket Mode Driver Library implements the MPLAB Harmony device driver and communicates with the WINC1500 Host Driver Software to access and control the external WINC1500 module which firmware consists of the following key features:

- Wi-Fi IEEE 802.11 b/g/n STA, AP, and Wi-Fi Direct® modes
- Wi-Fi Protected Setup (WPS)
- Support of WEP, WPA/WPA2 personal, and WPA/WPA2 Enterprise security
- Embedded network stack protocols
- Embedded TCP/IP stack with BSD-style socket API
- Embedded network protocols – DHCP client/server – DNS resolver client – SNTP client for UTC time synchronization
- Embedded TLS security abstracted behind BSD-style socket API
- HTTP Server for provisioning over AP mode
- 8 MB internal Flash memory with OTA firmware upgrade
- Low power consumption with different power saving modes
- SPI, I2C, and UART support

The WINC1500 Host Driver Software is a C library which provides the host MCU application with necessary APIs to perform necessary WLAN and socket operations. The architecture of the WINC1500 Host Driver Software which runs on the host MCU is shown below, and the components of the host driver are described in the following diagram.



WLAN Application Interface API

This module provides an interface to the application for all Wi-Fi operations and any non-IP related operations. This includes the following services:

- Wi-Fi STA management operations
- Wi-Fi Scan
- Wi-Fi Connection management (Connect, Disconnect, Connection status, etc.) – WPS activation/deactivation
- Wi-Fi AP enable/disable
- Wi-Fi Direct enable/disable
- Wi-Fi power save control API
- Wi-Fi monitoring (Sniffer) mode

This interface is defined in the file: m2m_wifi.h.

Socket API

This module provides the socket communication APIs that are mostly compliant with the well-known BSD sockets. To comply with the nature of MCU application environment, there are differences in API prototypes and in usage of some APIs between WINC1500 sockets and BSD sockets.

This interface is defined in the file: socket.h.

Host Interface (HIF)

The Host Interface is responsible for handling the communication between the host driver and the WINC1500 firmware. This includes interrupt handling, DMA and HIF command/response management. The host driver communicates with the firmware in a form of commands and responses formatted by the HIF layer.

The interface is defined in the file: m2m_hif.h.

Board Support Package (BSP)

The Board Support Package abstracts the functionality of a specific host MCU platform. This allows the driver to be portable to a wide range of hardware and hosts. Abstraction includes: pin assignment, power on/off sequence, reset sequence and peripheral definitions (Push buttons, LEDs…etc.).

The minimum required BSP functionality is defined in the file: nm_bsp.h.

Serial Bus Interface

The Serial Bus Interface module abstracts the hardware associated with implementing the bus between the Host and the WINC1500. The serial bus interface abstracts I2C, SPI, or UART bus interface. The basic bus access operations (Read and Write) are implemented in this module as appropriate for the interface type and the specific hardware.

The bus interface APIs are defined in the file: nm_bus_wrapper.h.

Using the WINC1500 Socket Mode Driver Library

The interface to the WINC1500 Socket Driver Library is defined in these header files:

- wdrv_winc1500_api.h
- wdrv_winc1500_stub.h

Any C language source (.c) file that uses the WINC1500 Socket Mode Driver library should include both wdrv_winc1500_api.h and wdrv_winc1500_stub.h.

Abstraction of the WINC1500 Wi-Fi Application

The major blocks of software comprise of a WINC1500 Wi-Fi application are listed and described in the following table.

Software Block	Description
Application Software	This is the application code. Note that three event handlers (OTA, Socket, and Wi-Fi) are part of this block. The event handlers contain callback functions that the driver calls and the application processes.
WINC1500 Socket Mode Driver	This is the WINC1500 Socket Mode driver. The driver API and Stub functions provide application software with setup for the SPI interface between the host MCU and the external WINC1500 module, and provide application software with control and socket data services to the external WINC1500 module via the WINC1500 Host Software Driver.
WINC1500 Socket Mode Driver Stub Functions	The driver Stub functions provide application software with control to the PIC32 MCU hardware blocks (SPI, EXT_INT, Timer, GPIO, I2C) for configuration of Host PIC32 MCU specific hardware and event handling: • SPI Interface • GPIO control • Timer • Interrupt from WINC1500 • Wi-Fi, TPC/IP socket, and OTA event handling
WINC1500 Socket Mode Driver API Functions	The driver Ext functions provide application software access to the driver's system interface for initialize and de-initialize of the driver, as well as setting up the driver hardware interrupt handler and the WINC1500 interrupt service routine. Note: Not all API functions are used in the socket mode driver library. Only these four API functions are used: • WDRV_EXT_Initialize • WDRV_EXT_Deinitialize • WDRV_EXT_HWInterruptHandler • WDRV_WINC1500_ISR
WINC1500 Host Software Driver APIs	The WINC1500 Host Software Driver provides the host MCU application with necessary APIs to perform necessary WLAN and BSD socket operations by offloading these operations to the WINC1500 module firmware. See Section 3 for details of these APIs.

Abstraction Model

This library provides a low-level abstraction of the WINC1500 Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The WINC1500 Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing
- Wi-Fi status
- Wi-FI console commands

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The Library Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the WINC1500 Socket Mode Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, hard interrupt handler, and interrupt service routine. See wdrv_winc1500_api.h, and wdrv_winc1500_stub.h.

Data Transfer Functions	Provides data transfer functions available in the configuration. See wdrv_winc1500_api.h.
Status Functions	Provides status functions. See m2m_wifi.h
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

This section describes how the WINC1500 Socket Mode Driver Library operates.

Description

The library provides host PIC32 MCU Wi-Fi application with interface support for the external WINC1500 module and offloads Wi-Fi and BSD socket operations to the WINC1500 module firmware.

Configuring the SPI Driver

This section describes the configuration settings for the WINC1500 Socket Mode Driver Library.

Description

Configuration

The WINC1500 hardware requires a specific configuration of the SPI driver to work correctly. Inside the MHC SPI driver configuration make sure to select:

- SPI clock rate of 8000000 or less
- Input phrase of SPI_INPUT_SAMPLING_PHASE_AT_END
- Clock mode of DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL

Recommended Settings

- Interrupt Driver mode
- Enhanced Buffer mode
- DMA mode enabled
- DMA Block Transfer Size to 512
- Size of DMA Buffer for dummy data to 512
- Ensure when setting up DMA in interrupt mode that the DMA interrupts are a higher priority than the SPI Driver interrupt

Examples

```
/*** SPI Driver Configuration ***/
#define DRV_SPI_NUMBER_OF_MODULES 6
/*** Driver Compilation and static configuration options. ***/
/*** Select SPI compilation units.***/
#define DRV_SPI_POLLED 0
#define DRV_SPI_ISR 1
#define DRV_SPI_MASTER 1
#define DRV_SPI_SLAVE 0
#define DRV_SPI_RM 1
#define DRV_SPI_EBM 0
#define DRV_SPI_8BIT 1
#define DRV_SPI_16BIT 0
#define DRV_SPI_32BIT 0
#define DRV_SPI_DMA 1
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER 1
#define DRV_SPI_CLIENTS_NUMBER 1
#define DRV_SPI_ELEMENTS_PER_QUEUE 10
/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE 512
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE 512
/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0 SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0 DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0 DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0 false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0 DRV_SPI_PROTOCOL_TYPE_STANDARD
```

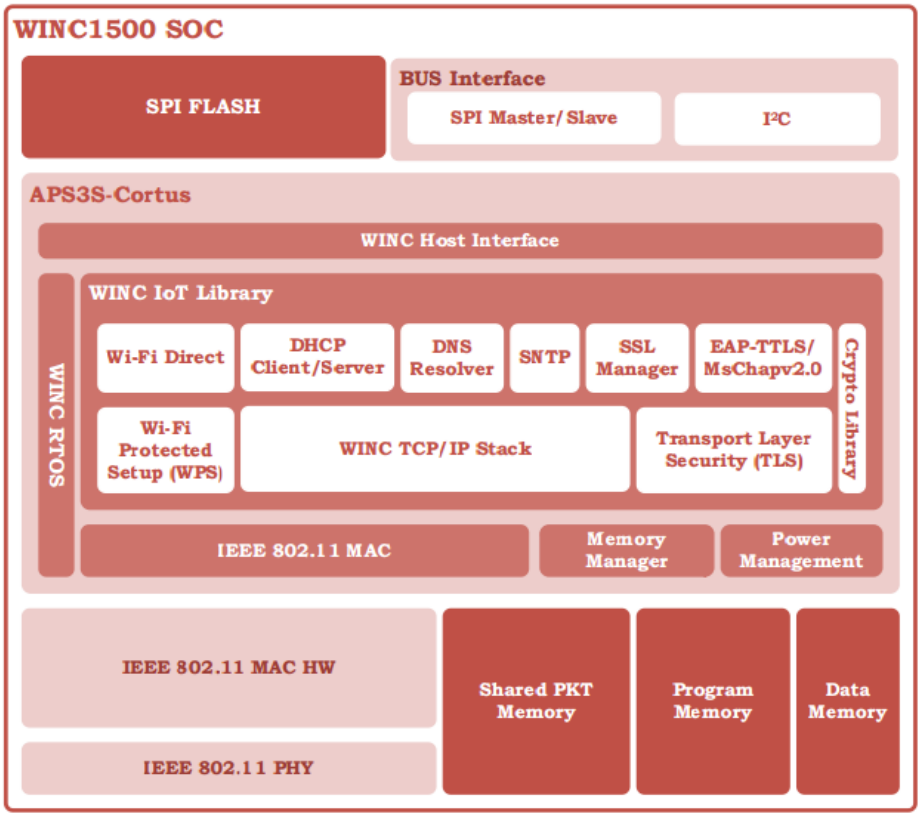
```
#define DRV_SPI_COMM_WIDTH_IDX0 SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_CLOCK_SOURCE_IDX0 SPI_BAUD_RATE_PBCLK_CLOCK
#define DRV_SPI_SPI_CLOCK_IDX0 CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0 8000000
#define DRV_SPI_BUFFER_TYPE_IDX0 DRV_SPI_BUFFER_TYPE_STANDARD
#define DRV_SPI_CLOCK_MODE_IDX0 DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0 SPI_INPUT_SAMPLING_PHASE_AT_END
#define DRV_SPI_TRANSMIT_DUMMY_BYTE_VALUE_IDX0 0x00
#define DRV_SPI_TX_INT_SOURCE_IDX0 INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0 INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0 INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_TX_INT_VECTOR_IDX0 INT_VECTOR_SPI1_TX
#define DRV_SPI_RX_INT_VECTOR_IDX0 INT_VECTOR_SPI1_RX
#define DRV_DRV_SPI_ERROR_INT_VECTOR_IDX0 INT_VECTOR_SPI1_FAULT
#define DRV_SPI_TX_INT_PRIORITY_IDX0 INT_PRIORITY_LEVEL1
#define DRV_SPI_TX_INT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_RX_INT_PRIORITY_IDX0 INT_PRIORITY_LEVEL1
#define DRV_SPI_RX_INT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_ERROR_INT_PRIORITY_IDX0 INT_PRIORITY_LEVEL1
#define DRV_SPI_ERROR_INT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0 10
#define DRV_SPI_RESERVED_JOB_IDX0 1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0 DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0 16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0 DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0 16
```

WINC1500 Module Firmware Overview

Provides an overview of the firmware for the WINC1500 Module.

Description

The firmware comprises the Wi-Fi IEEE-802.11 MAC layer and embedded protocol stacks which offload the host MCU. The components of the system are described in the following sub-sections.



WLAN APIs

This WLAN APIs provide an interface to the application for all Wi-Fi operations and any non-IP related operations. This includes the following

services:

- Wi-Fi STA management operations
- Wi-Fi Scan
- Wi-Fi Connection management (Connect, Disconnect, Connection status, etc.)
- WPS activation/deactivation
- Wi-Fi AP enable/disable
- Wi-Fi Direct enable/disable
- Wi-Fi power save control API
- Wi-Fi monitoring (Sniffer) mode

This interface is defined in the file: `m2m_wifi.h`.

Socket API

The socket APIs are mostly compliant with the BSD sockets and to comply with the nature of MCU application environment, there are differences in API prototypes and in usage of some APIs between WINC1500 sockets and BSD sockets.

This interface is defined in the file: `socket.h`.

IoT Library

The IoT library provides a set of networking protocols in WINC1500 firmware. It offloads the host MCU from networking and transport layer protocols. The following sections describe the components of WINC1500 IoT library.

- WINC1500 TCP/IP STACK - The WINC TCP/IP is an IPv4.0 stack based on the uIP TCP/IP stack (pronounced micro IP).
- DHCP CLIENT/SERVER - A DHCP client is embedded in WINC1500 firmware that can obtain an IP configuration automatically after connecting to a Wi-Fi network. WINC1500 firmware provides an instance of a DHCP server that starts automatically when WINC AP mode is enabled. When the host MCU application activates the AP mode, it is allowed to configure the DHCP Server IP address pool range within the AP configuration parameters.
- DNS RESOLVER – WINC1500 firmware contains an instance of an embedded DNS resolver. This module can return an IP address by resolving the host domain names supplied with the socket API call gethostbyname.
- SNTP - The SNTP (Simple Network Time Protocol) module implements an SNTP client used to synchronize the WINC1500 internal clock to the UTC clock.
- EAP-TTLS/MSCHAPV2.0 - This module implements the authentication protocol EAP-TTLS/MsChapv2.0 used for establishing a Wi-Fi connection with an AP by with WPA-Enterprise security.
- TRANSPORT LAYER SECURITY - For TLS implementation.
- WI-FI PROTECTED SETUP - For WPS protocol implementation.
- WI-FI DIRECT - For Wi-Fi Direct protocol implementation.
- CRYPTO LIBRARY - The Crypto Library contains a set of cryptographic algorithms used by common security protocols. This library has an implementation of the following algorithms:
- MD4 - Hash algorithm (Used only for MsChapv2.0 digest calculation)
- MD5 - Hash algorithm
- SHA-1 - Hash algorithm
- SHA-256 - Hash algorithm
- DES Encryption (Used only for MsChapv2.0 digest calculation)
- MS-CHAPv2.0 (Used as the EAP-TTLS inner authentication algorithm)
- AES-128, AES-256 Encryption (Used for securing WPS and TLS traffic)
- BigInt module for large integer arithmetic (for Public Key Cryptographic computations)
- RSA Public Key cryptography algorithms (includes RSA Signature and RSA Encryption algorithms)

Host Interface Driver Wi-Fi Events

Provides information on the Host Interface Driver Wi-Fi events.

Description

There are four categories of events:

• Wi-Fi events

• Socket events

• OTA (Over-The-Air) update events

• Error Events

Wi-Fi Events

Wi-Fi events must be customized to suit the application. The WINC1500 socket driver calls the event callback function to notify the application of Wi-Fi events. The p_eventData parameter points to a ‘C’ union of containing all possible Wi-Fi event data. Not all events have data associated with them – in this case the pointer will be NULL. When an event occurs, the event data should be read as soon as possible before another event

occurs which will overwrite data from the previous event.

If the event data is to be retrieved outside the event handler function, the utility function m2m_wifi_get_wifi_event_data() returns a pointer to the t_wifiEventData union.

Socket Events

Socket events are handled, but must be customized to suit the application. The WINC1500 driver calls the socket event callback function to notify the application of socket events. The p_eventData parameter points to a 'C' union of containing all possible socket event data. Not all events have data associated with them – in this case the pointer will be NULL. When an event occurs, the event data should be read as soon as possible before another event occurs which will overwrite data from the previous event.

If the event data is to be retrieved outside the event handler function, the utility function m2m_wifi_get_socket_event_data() returns a pointer to the t_socketEventData union.

OTA Events

OTA events are associated with downloading and switching to a new WINC1500 firmware image downloaded via the Wi-Fi network. The WINC1500 driver calls the OTA event callback function to notify the application of OTA events. The p_eventData parameter points to a 'C' structure containing the OTA event data.

If the event data is to be retrieved outside the event handler function, the utility function m2m_wifi_get_ota_event_data() returns a pointer to the t_otaEventData structure.

Error Events

The application is notified of error events via the callback. Error codes are defined in: `wf_errors.h`.

Configuring the Library

This section describes how to configure the WINC1500 Wi-Fi driver.

Description

The configuration of the WINC1500 Wi-Fi Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the WINC1500 Socket Mode Driver Library. Based on the selection, the WINC1500 Socket Mode Driver Library may support the selected features. These configuration settings will apply to all instances of the WINC1500 Socket Mode Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the WINC1500 Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/winc1500`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
wdrv_winc1500_stub.h	Contains Stub function prototypes for interfacing to the WINC1500 Wi-Fi Driver.
wdrv_winc1500_api.h	Contains API function prototypes for interfacing to the WINC1500 Wi-Fi Driver.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source Folder Name	Description
/driver/wifi/winc1500/dev/console/wdrv_winc1500_console.c	Console module for WINC1500 wireless driver.
/driver/wifi/winc1500/dev/gpio/wdrv_winc1500_eint.c	External interrupt handler for WINC1500 wireless driver.

/driver/wifi/winc1500/dev/gpio/wdrv_winc1500_gpio.c	GPIO interface for WINC1500 wireless driver.
/driver/wifi/winc1500/dev/spi/wdrv_winc1500_spi.c	Support SPI communications to the WINC1500 module.
/driver/wifi/winc1500/dev/timer/wdrv_winc1500_timer.c	Timer functions for WINC1500 wireless driver.
/driver/wifi/winc1500/osal/wdrv_winc1500_osal.c	OS abstraction layer for WINC1500 wireless driver.
/driver/wifi/winc1500/wireless_driver_extension/common/source/nm_common.c	This module contains common APIs implementations.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/m2m_hif.c	This module contains M2M host interface API implementations.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/m2m_ota.c	WINC1500 IoT OTA Interface.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/m2m_periph.c	WINC1500 Peripherals Application Interface.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/m2m_wifi.c	This module contains M2M Wi-Fi APIs implementation.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/nmasic.c	This module contains WINC1500 ASIC specific internal APIs.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/nmbus.c	This module contains WINC1500 bus APIs implementation.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/nmdrv.c	This module contains WINC1500 M2M driver APIs implementation.
/driver/wifi/winc1500/wireless_driver_extension/driver/source/nmspi.c	This module contains WINC1500 SPI protocol bus APIs implementation.
/driver/wifi/winc1500/wireless_driver_extension/socket/source/socket.c	WINC1500 BSD Compatible Socket Interface.
/driver/wifi/winc1500/wireless_driver_extension/spi_flash/source/spi_flash.c	WINC1500 SPI flash interface.
/driver/wifi/winc1500/wireless_driver_extension/wdrvext_winc1500.c	WINC1500 wireless driver extension.
/driver/wifi/winc1500/wireless_driver_extension/winc1500_fw_update.c	WINC1500 firmware update support.
/driver/wifi/winc1500/wireless_driver_extension/winc1500_task.c	Entry point of WINC1500 core driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	The WINC1500 Wi-Fi Driver controller has no optional files.

Module Dependencies

The WINC1500 Socket Mode Driver Library depends on the following modules:

- SPI Driver Library
- SPI Flash Driver Library
- WINC1500 Wi-Fi Driver Ethernet Mode Library
- Timer Driver Library
- USART Driver Library

Library Interface

This section describes the Application Programming Interface (API) functions of the WINC1500 Socket Mode Driver. Refer to the *WINC1500 Wi-Fi Driver Library* > Library Interface section for information on the APIs in that library that are also used by the WINC1500 Socket Mode Driver.

Refer to each section for a detailed description.

WINC1500 Firmware Update Utility

Refer to WINC1500 Firmware Update Guide for detailed information on using the firmware update utility.

Encoder Libraries Help

This section provides descriptions of the software Encoder libraries that are available in MPLAB Harmony.

Opus Encoder Library

This section describes the Opus Encoder Library, which utilizes the same prebuilt library as the Opus Decoder.

Introduction

Introduces the Opus Encoder Library.

Description

Opus is an open, royalty-free, highly versatile audio codec. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716, which incorporated technology from Skype's SILK codec and Xiph.Org's CELT codec.

Opus Features

Supported features are:

- Bit rates from 6 kb/s to 510 kb/s
- Sampling rates from 8 kHz (narrowband) to 48 kHz (fullband)
- Frame sizes from 2.5 ms to 60 ms
- Support for both constant bit rate (CBR) and variable bit rate (VBR)
- Audio bandwidth from narrowband to fullband
- Support for speech and music
- Support for mono and stereo
- Support for up to 255 channels (mult-istream frames)
- Dynamically adjustable bit rate, audio bandwidth, and frame size
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

The full specification, RFC 6716, including the reference implementation is available from http://www.opus-codec.org. An up-to-date implementation of the Opus standard is available from the Opus Codec downloads page by visiting: https://www.opus-codec.org/downloads/

Typical Applications

- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies
- Toys
- Robots
- Any application using message playback

Resources

Feature	Option	Usage
Opus Encoder Library Flash Size	Release Build	292 KB
Opus Encoder RAM	Heap Size (Stereo mode, 16 kHz, 16-bit Audio)	31 KB

Using the Library

This topic describes the basic architecture of the Opus Encoder Library and provides information and examples on its use.

Description

Interface Header File: `opus_enc.h`

The interface to the Opus Encoder Library is defined in the `opus.h` header file. Any C language source (`.c`) file that uses the Opus Encoder Library should include `opus_enc.h`.

Please refer to the What is MPLAB Harmony? section for how the Opus Encoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Opus Encoder Library module.

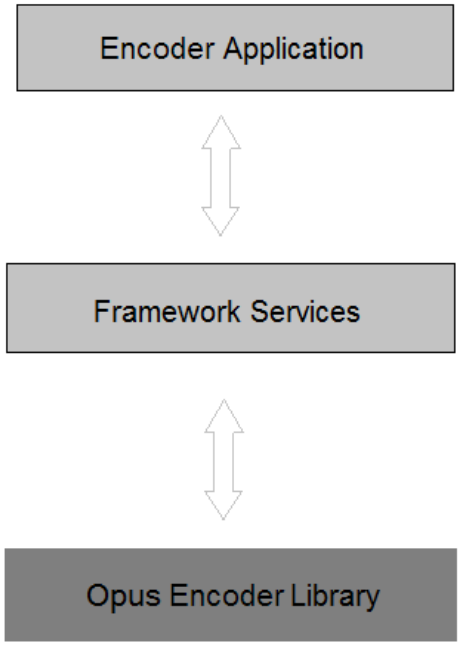
Abstraction Model

Describes the abstraction model for the Opus Encoder Library.

Description

This Opus Library is an Open Source/Free Software patent-free audio compression format designed for interactive speech and music transmission over the Internet. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model



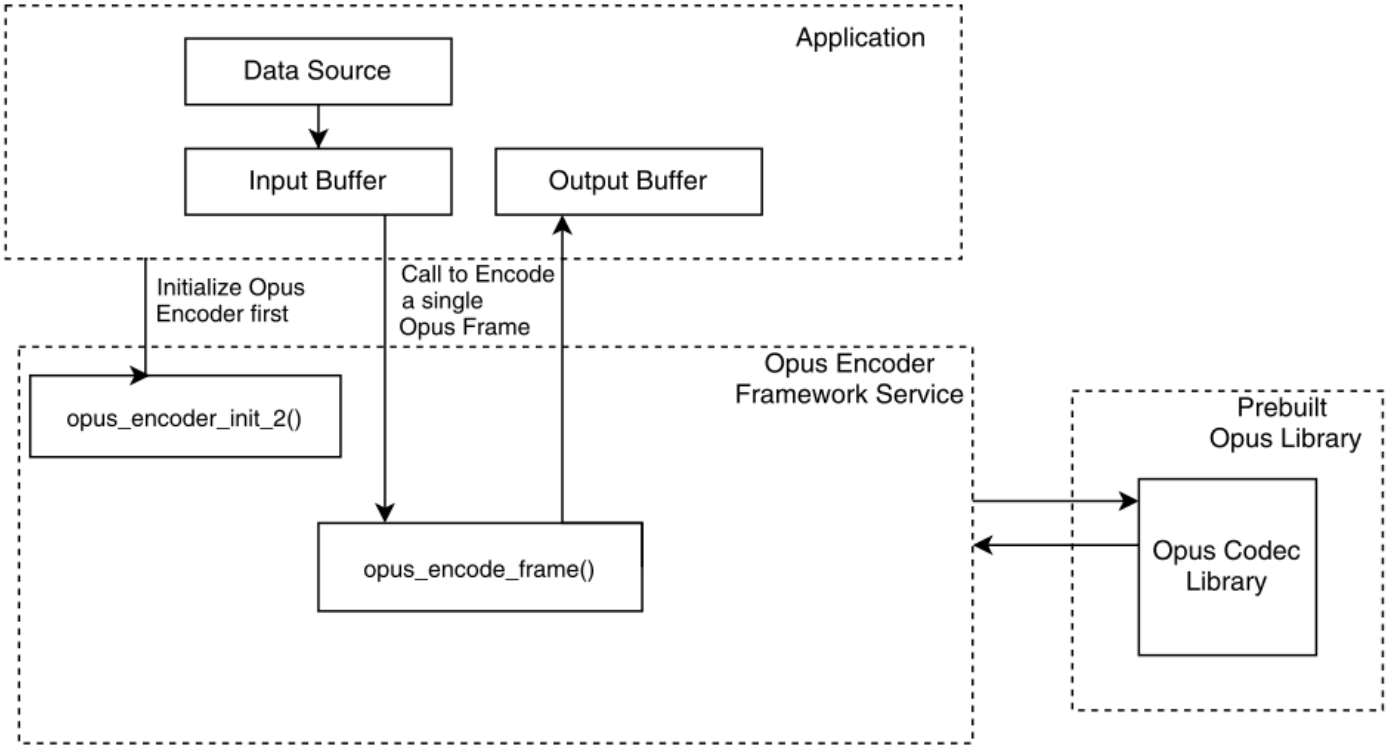
How the Library Works

This section describes how to work with the MPLAB Harmony Opus abstraction layer interface, which provides simplifier APIs for using the Opus Encoder Library.

Description

For complete documentation and how to implement Opus, visit https://www.opus-codec.org.

The following diagram describes the data flow between the Opus Encoder and an application.



Initializing the Opus Encoder

Initialize the Opus Encoder by calling the function opus_encoder_init_2 from the `opus_enc.h` file. This function initializes the Opus Encoder state structure. The application can start to decode Opus frame after initialization.

Decoding an Opus Frame

The opus_encode_frame function in `opus_enc.h` is used to encode a single Opus frame, and write back encoded data into an output buffer.

Code Example

The following code provides an example for initializing the Opus Encoder and encoding a packet.

```
bool encoder_init(EncoderType encoder_type)
{
    // initialize encoder state
    bool ret = false;
    encoder_state.encoder_type = encoder_type;
    ret = opus_encoder_init_2(stream_info.channel, stream_info.sample_rate);
    return ret;
}

void App_Task()
{
    while(1){
        while(not the last packet of audio data){

            // Encode one packet
            bool ret = opus_encode_frame(pin, insize, pout, outsize);
            if(ret == true)
            {
                // continue decoding
            }else{
                // handle decoding errors;
            }

        }
        // Clean up
        opus_encoder_free();
        break;
    }
}
```

Library Interface

This section describes the Application Programming Interface (API) functions of the Opus Encoder Library.

Refer to each section for a detailed description.

a) General Functions

b) Data Types and Constants

Files

This section lists the source and header files used by the Opus Encoder Library.

Speex Encoder Library

This section describes the Speex Encoder Library, which utilizes the same prebuilt library as the Speex Encoder.

Introduction

This section introduces the Speex Encoder Library.

Speex is an open source software audio compression format designed for speech. The Speex Project aims to lower the barrier of entry for voice applications by providing a free alternative to expensive proprietary speech codecs. Speex is well-adapted to Internet applications and provides useful features that are not present in most other codecs. Finally, Speex is part of the GNU Project and is available under the revised BSD license.

Speex Features

Supported features are:

- Narrowband (8 kHz), wideband (16 kHz), and ultra-wideband (32 kHz) compression in the same bitstream
- Intensity stereo encoding
- Packet loss concealment
- Variable bitrate operation (VBR)
- Voice Activity Detection (VAD)
- Discontinuous Transmission (DTX)
- Fixed-point port
- Acoustic echo canceller
- Noise suppression

Feature	Option	Usage
Speex Encoder Library Flash Size	Release Build	20KB
Speex Encoder RAM	Heap Size (narrow band)	33KB ~ 35KBs

Using the Library

This topic describes the basic architecture of the Speex Encoder Library, and provides information and examples on its use.

Description

The interface to the Speex Encoder Library is defined in the `speex_enc.h` header file. Any C language source (`.c`) file that uses the Speex Encoder Library should include `speex _enc.h`.

Please refer to the What is MPLAB Harmony? section for how the Speex Encoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks, or the overall operation of the

Speex Encoder Library module.

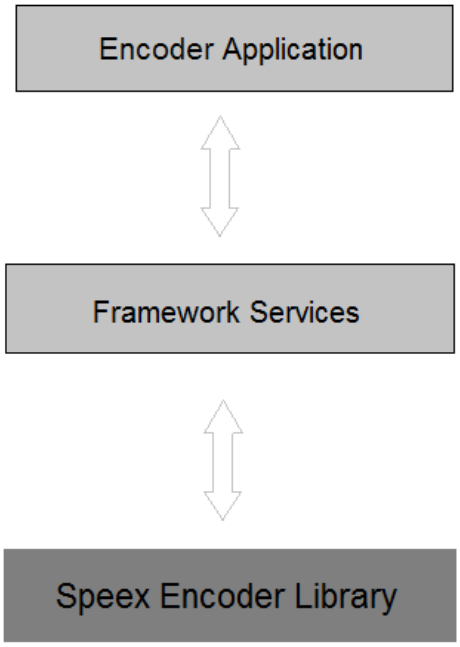
Abstraction Model

This section describes the abstraction model for the Speex Encoder Library.

Description

This Speex Library is an open source, patent-free audio compression format designed speech. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_encoders demonstration for reference.

Abstraction Model



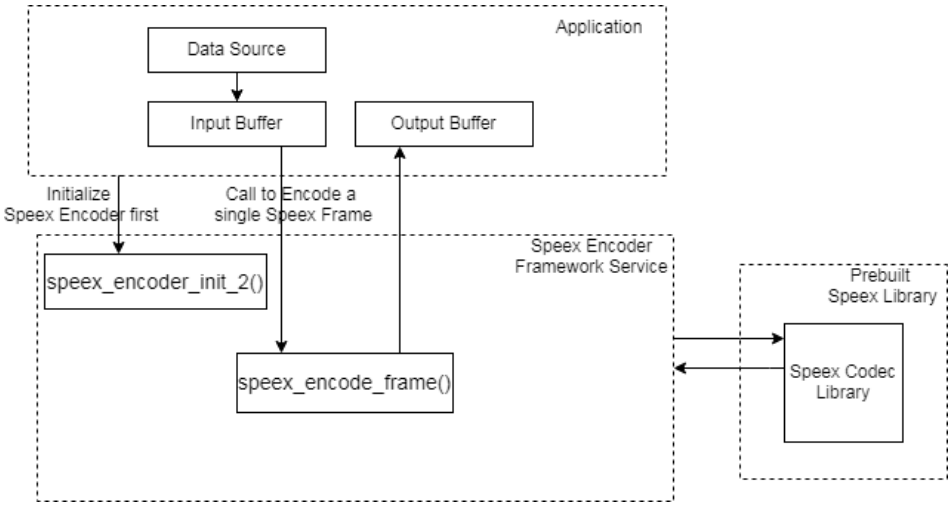
How the Library Works

This section describes how to work with the MPLAB Harmony Speex Encoder abstraction layer interface, which provides simplified APIs for using the Speex Encoder Library.

Description

For complete documentation and how to implement Speex, visit https://www.speex.org.

The following diagram describes the data flow between the Speex Encoder and an application.



Initializing the Speex Encoder

Initialize the Speex Encoder by calling the function `speex_encoder_init_2` from the `speex_enc.h` file. This function initializes the Speex Encoder state structure. The application can start to decode the Speex frame after initialization.

Decoding a Speex Frame

The `speex_encode_frame` function in `speex_enc.h` is used to encode a single Speex frame, and write back encoded data into an output buffer.

Code Example

The following code provides an example for initializing the Speex Encoder and encoding a packet.

```
bool encoder_init(EncoderType encoder_type)
{
// initialize encoder state
bool ret = false;
encoder_state.encoder_type = encoder_type;
ret = speex_encoder_init_2(stream_info.channel, stream_info.sample_rate);
return ret;
}
void App_Task()
{
while(1){
while(not the last packet of audio data){
// Encode one packet
bool ret = opus_encode_frame(pin, insize, pout, outsize);
if(ret == true)
{
// continue decoding
}else{
// handle decoding errors;
}
}
// Clean up
opus_encoder_free();
break;
}
}
```

Library Interface

This section describes the Application Programming Interface (API) functions of the Speex Encoder Library.

Refer to each section for a detailed description.

a) General Functions

b) Data Types and Constants

Files

This section lists the source and header files used by the Speex Encoder Library.

Graphics Libraries Help

This topic provides information about the graphics libraries that are available in MPLAB Harmony.

Currently, MPLAB Harmony provides two solutions for developing graphics firmware:

- The MPLAB Harmony Graphics Composer (MHGC) Suite - The MPLAB Harmony Graphics Composer Suite (MHGC) is a free, modular graphics stack and tools suite for use with Microchip PIC32 microcontrollers. The MHGC tool provides an easy to use GUI that works within the MPLAB X IDE environment.
- The SEGGER emWin Graphics Library - In addition to the standard Graphics Library, the MPLAB Harmony Integrated Software Framework also offers a third-party graphics library, emWin, from SEGGER Microcontroller GmbH & Co. KG. The SEGGER emWin Graphics Library provides an efficient, processor and LCD controller-independent Graphical User Interface (GUI) for applications that operate with a graphical LCD.

MPLAB Harmony Graphics Composer (MHGC) Suite

This section describes the MPLAB Harmony Graphics Composer (MHGC) Suite.

Introduction

This section provides an overview of the MPLAB Harmony Graphics Composer (MHGC) Suite.

Description

The MPLAB Harmony Graphics Composer (MHGC) Suite is a free, modular graphics stack and tools suite for use with Microchip PIC32 microcontrollers. The MHGC suite provides an easy to use GUI that works within the MPLAB X IDE environment. This is tightly coupled with MPLAB Harmony Configurator (MHC), code development, and other integrated debug features. The tools provide a simplified interface to create graphics content, target specific processor / display and touch interface hardware, and generate code. In most cases, no additional programming to support graphics is required at all, which reduces development time. For more information about the MHGC, refer to MPLAB Harmony Graphics Composer User's Guide.

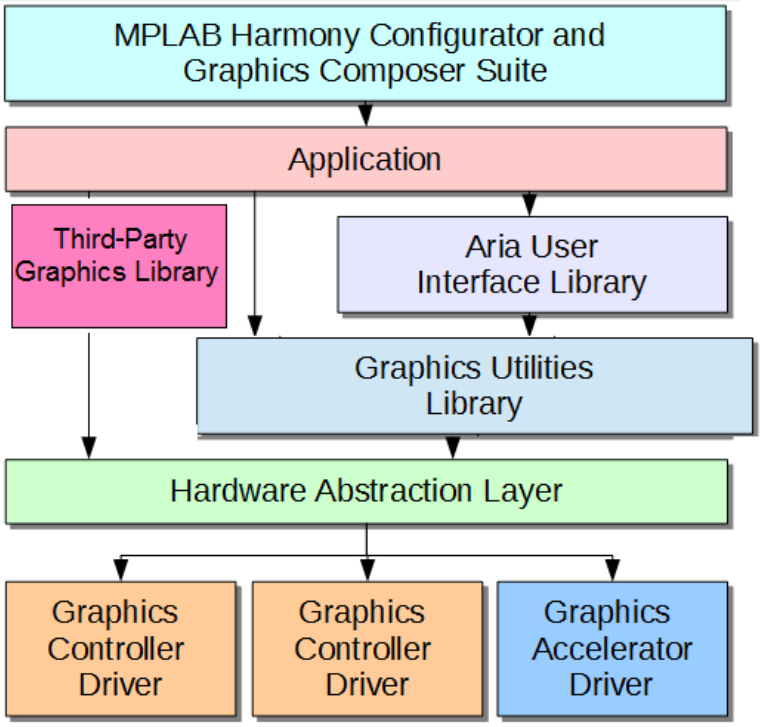
The MPLAB Harmony Graphics Stack consists of several layers that directly build on the capabilities of lower layers to provide a robust framework for displaying rich graphics on supported display devices. Higher layers can be removed as needed if their capabilities are not required.

Graphics Stack Architecture

This section describes the graphics stack architecture.

Description

The following is a diagram of the graphics stack architecture.



The functionality of each layer is summarized below, the details have been provided in the help document in the corresponding sections:

- Graphics Controller Driver – Software that talks directly to hardware. Multiple drivers for internal, external and no-controller options are available. These can be customized with the new Display Manager interface. No other software in the stack should have hardware access.
- Graphics Accelerator Driver- Software that interfaces with graphics accelerator hardware.
- Hardware Abstraction Layer (HAL) – A software layer that serves as a gate-keeper for all graphics controller and accelerator drivers. This layer is configured at initialization by the underlying graphics drivers and provides functionality such as: buffer management, primitive shape drawing, hardware abstraction, and draw state management. The presence of this layer serves as a means of protection for the drivers, frame buffers, and draw state in order to prevent state mismanagement by the application.
- Graphics Utilities Library – This library is primarily responsible for managing and decoding assets such as images, fonts, and strings. It provides the means for interacting with asset data, complex data decoding, data decompression, and string asset look-up. It also abstractly handles accessing external memory sources during asset decoding.
- Aria User Interface Library – This library provides the capability for interface generation, management, and interaction. This library provides the building blocks for constructing a user interface in the form of “Widgets” or user interface elements. These consist of things like buttons, check-boxes, images, etc. This library also handles user interaction events for things like touch actions.
- Third Party Graphics Library – The third party library can be used with the harmony framework to perform the graphics operations if desired by the user. The third party library has access to the hardware abstraction layer (HAL), which has been configured to supply the frame buffer to be filled in by the third-party graphics library.
- MPLAB Harmony Graphics Composer (MHGC) – This tools suite provides the capability to design a user interface using a graphical drag and drop interface. The tool can write all of the code needed to initialize, configure, and manage an Aria library context. The tools include:
- Graphics Asset Converter – new engine for importing multiple external image types, estimating and optimizing size,
- Image Editor - Enabling palette, compression, format changes and editing of images without external tools
- Resource Manager – Tabulated totals of memory usage for images, fonts and other elements used within the graphics design. These can be used to optimize a specific design to fit within a given device Flash memory.
- WYSIWYG GUI editor – Enables drag and drop capability to visualize your particular design
- Event Manager – Enables the user to customize the experience of touch and logical (application) events and to interact with graphical attributes
- Tree Manager – Enables the user to select the drawing priority and establish parent / child relationships so that objects can be grouped as desired
- String and Font Manager – Used to input strings in multiple languages for potential reuse, and optimization of fonts and memory requirements

The Graphics Library architecture components, display drivers, libAria APIs and demonstration applications are placed into the following locations during the installation of MPLAB Harmony:

- `/microchip/harmony/<version>/framework/gfx/hal`
- `/microchip/harmony/<version>/framework/gfx/utils`
- `/microchip/harmony/<version>/framework/gfx/libaria`
- `/microchip/harmony/<version>/apps/gfx/`

Graphics Composer Suite Salient Features

This section provides the highlights of the new MPLAB Harmony Graphics Composer Suite.

Description

Multiple new features have been added to the tools starting with the introduction of MPLAB Harmony v2.02.

These tools were reworked based on:

- New hardware capabilities from Microchip PIC32 products
- Numerous customer requests for updated features.

The tools include:

- New graphics import engine (GAC)
- Capability of image conversion and compression
- Hardware Abstraction Layer (HAL), support for GPU
- Entirely new graphics library (Aria User Interface Library)
- Localization font and string manager; font filtering
- Resource utilization manager
- 24-bit color (32-bit with alpha channel) and multi-layer support
- Multiple new widgets, support for primitive touch gestures
- Multiple new applications to demonstrate features (old apps will be retired)
- Tree drawing support, parent child association
- Revised WYSIWYG engine, updated accuracy and screen elements
- Revised clipping and object drawing support
- New interface for graphics events, and external / logical macros
- Integration of the Display Manager for automatic generation of display drivers

In addition to the rich features offered on introduction, the new architecture enables new capabilities that are planned in future releases.

These new features include:

- Integration of the PIC32MZ DA LCD driver (GLCD)
- Support for the PIC32MZ DA GPU library
- Mechanism for the motion/movement engine
- Mechanism for the simulation engine
- Editor/user customization of widgets
- Integrated image editor
- Touch screen gesture editing
- Memory-saving global pallet functions, optimized use of SRAM buffers

Graphics Composer Suite Goals

This section describes the MPLAB Harmony Graphics Composer Suite goals.

Description

The graphics tools and stack were designed with several goals in mind:

- Tight Integration Experience – Design and code generator tools are tightly integrated with the development environment for one-touch project generation
- User Experience – Libraries and tools are easy to learn and use
- Powerful User Interface Library – User interface library builds upon and expands previous capabilities to offer increased functionality
- Complete Code Generation – Can generate code for library initialization, library management, touch integration, color schemes, event handling, and screen macros
- Powerful Asset Converter – Can output several image formats, performs auto palette generation for image compression, supports run-length encoding, and supports several popular image asset formats. Also supports automatic font character inclusion and rasterization.
- Expanded Color Mode Support – The stack can manage frame buffers using 8-bit to 32-bit color
- Enhanced Resource Configuration – Tools provide the capability to manage assets more completely
- Text localization – The stack provides the capability to easily integrate international language characters into a design and seamlessly change between defined languages at run-time
- Abstract Hardware Support – Graphics controllers and accelerators should be able to be added or removed without any change to the application
- Enable future features – Including simulator and motion functions

Graphics Composer Porting

This section describes the MPLAB Harmony Graphics Composer porting process.

Description

Within MPLAB Harmony releases earlier than v2.02, the MPLAB Harmony Graphics Composer tool offered only a subset of the features currently provided. This feature subset included tools that were removed in the current release. These removed tools include:

- Graphics Object Library (GOL)
- Graphic Asset Converter (GAC)

Customers who must have legacy access to these tools can still find them in a parallel version of MPLAB Harmony. These tools will continue to be available within the v1.xx versions for some time.

- On update of the tools, any designs previously constructed using graphics will be offered an update. This process will port the old design to the new structure. While the process is automatic, it may take some engineering to refine items like widget compatibility. Most objects, their events, parameters and locations will be ported into the new design without further intervention.

Aria User Interface Library

This section describes the Aria User Interface for the graphics capabilities and operations.

Description

Introduction

Introduces the Aria User Interface Library.

Description

The Aria User Interface Library is primarily responsible for presenting a visual means of interaction between a user and an application. The library provides the building blocks to construct a complex user interface and is responsible for managing the interface once created. It is also responsible for responding to external input from users other sources and reacting appropriately. The goals of this library are to be:

- Able to provide a simple but powerful user experience
- Customizable to the needs of the application
- Light and flexible with regards to resource consumption
- Easily extensible to meet future design needs

Definitions

Alignment – Indicates the placement of objects within a given bounding area

Bounding Rectangle – The rectangle that an object occupies in a given space

Context – A discrete instance of the user interface library

Event – An indication of some kind of occurrence that may require attention

Layer – Directly related to the layers offered by the Hardware Abstraction Layer. Aria layers also function as direct children to a screen. Widgets are added to layers and become part of the overall widget tree.

Margin – A buffer area at the edge of a bounding rectangle

Occlusion – The state of being completely obstructed by another entity.

Rasterize – The process of translating a user interface model from a logical mathematical representation into a visual image.

Scheme – A list of colors that can be referenced for drawing purposes

Screen – The root node of a widget tree. Represents a discrete configuration of layers and widgets. Can have a unique life cycle for custom memory management.

String – A logical array of linguistic characters

Widget – An abstract object that is part of a user interface

Widget Tree – A tree data structure of widgets that, when rendered, generates a user interface image.

Overview

The Aria user interface library is primarily responsible for:

- HAL Configuration
- Widget Tree Management
- Event Management
- Input Handling

- Scene Rendering

HAL Configuration

The Aria User Interface library is context-based similar to other portions of the graphics stack. For ease of use, the library is responsible for creating and managing a HAL context internally. This releases the application from having to interact with the HAL API at all.

The context contains all of the information required to manage the state of the library. It contains the screen state, the event list, the input state, and various other settings.

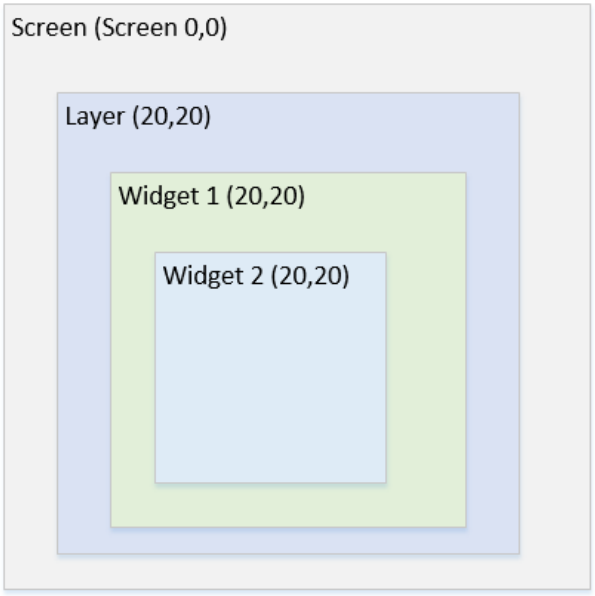
Widget Tree Management

The widget tree is a tree data structure comprised primarily of widgets. At its root is a screen object. Each of the screen object's direct children is a layer object. Any descendants of a layer are widgets.

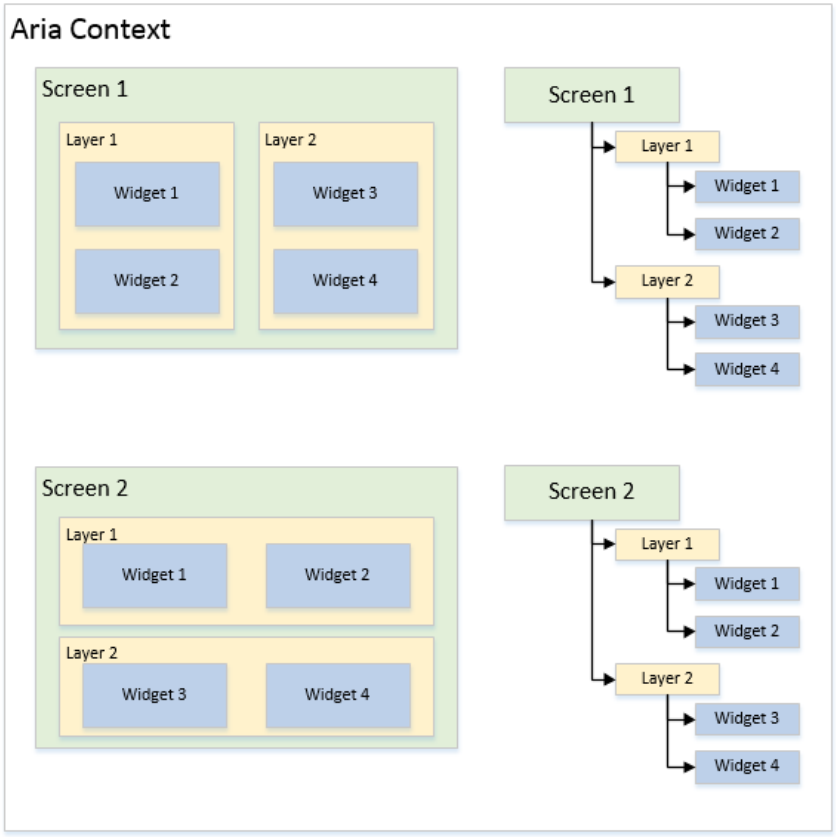
Heterogeneous Space

When dealing with objects in a tree it is helpful to understand objects do not live in the same coordinate space as their ancestors or descendants. Each level of the tree represents a unique area of spatial coordinates with the root coordinate space, or the screen space, being the physical space of the display device. Each space is a two dimensional Cartesian coordinate system in both the positive and negative directions.

For example, assume a widget is a child of a layer which is a child of a screen. The screen position is (0,0) in physical space. The layer position is (20,20). Widget 1 is at (20,20) and widget 2 is at (20,20). All of these coordinates aside from the screen are relative. Each widget is (20,20) offset from its parent. However, Widget 2 is not at (20,20) in physical space, it's actually (60,60). In global space each space builds on its parent, but it's entirely relative.



The following is two screen examples showing different visual representations with identical logical (tree) representations.



Using a tree to manage the logical state of a user interface provides numerous benefits.

- **Position Inheritance** – Child coordinate systems logically inherit from their parents but are not directly affected by them. Thus, if some ancestor changes position in its space, children likewise change overall position but their positions in their relative space do not change. This allows for easy manipulation of large portions of the user interface through very small changes.
- **Intelligent Rendering** – The tree structure allows for fast analysis of widget states when determining how and when to render the overall scene. Nodes in the tree track changes in the states of widgets and their descendants. This allows the library to use intelligent methods to cull portions of the user interface to avoid the processing overhead of redrawing widgets that have not changed.
- **Effect Propagation** – As with coordinate propagation, effects can also be inherited along tree branches. For instance, if some node in the tree is made invisible, all descendants are also made invisible. If a node is made partially transparent, then that transparency is propagated to all descendant nodes.

Screens

A screen is defined as the logical root of the user interface scene. Its direct descendants are always one or more layers, as seen in the above diagram. Its size always matches the physical dimensions of the display device used.

Life Cycle

The life cycle of a screen can be configured to better manage run-time memory usage. The relevant options are:

Persistent – By default screens will create their widget tree when shown and will free the memory consumed by their widget tree when they are hidden. A screen that is marked as persistent will not free their widget tree when hidden. This allows widgets in a screen to maintain their state when the screen is no longer visible. The downside is that more heap memory is consumed.

Create At Startup – By default screens are only created when they are shown thus keeping run-time memory usage to a minimum. However, the application may want to access widgets in a screen before it is shown for the first time. This option will cause the screen to allocate all of its memory when the screen is first added to the user interface library context.

Orientation

Because the Hardware Abstraction Layer supports dynamic orthogonal orientation, screens can take advantage of this feature. Thus, screens have the option to set a magnitude of rotation to some factor of 90 degrees.

Layers

User interface layers serve several functions. They function as the de-facto root parent for widgets, they directly configure hardware layers in the graphics driver, and they manage per-layer effects.

In the simplest case the hardware supports a single layer that is the same dimension as the physical display. More advanced cases may have several layers that can have unique coordinates and dimensions within the physical display space.

When a screen becomes active, it iterates over all of its layers and applies the settings of each through the appropriate HAL APIs to set up the

display state for that particular screen. Two screens may have different layer counts, layouts, and buffer counts.

Schemes

Schemes in Aria are simply collections of colors with given names. If a scheme is assigned to a widget then that scheme will be referenced by that widget during rendering. Aria has an internal scheme that all widgets use by default in the event that a scheme is not assigned.

Below is a list of scheme colors and a description of how it is often used. There is no restriction on how a widget references a scheme. The color names are merely a recommendation.

Scheme Colors

- **Base** – Default area fill
- **Highlight** – Light embossing
- **Highlight Light** – Very light embossing
- **Shadow** – Dark embossing
- **Shadow Dark** – Very dark embossing
- **Foreground** – A foreground color
- **Foreground Inactive** – Foreground color when inactive
- **Foreground Disabled** – Foreground color when disabled
- **Background** – A background color, usually to differentiate from Base
- **Background Inactive** – Background when inactive
- **Background Disabled** – Background when disabled
- **Text** – Text color
- **Text Highlight** – Text color background when highlighted
- **Text Highlight Text** – Text color when highlighted
- **Text Inactive** – Text color when inactive
- **Text Disabled** – Text color when disabled

Widgets

A widget is an abstract representation of an object in the user interface. In its most basic form it is a rectangle that is capable of drawing a border, a background color, and containing child widgets. More specific implementations extend the basic widget implementation to provide advanced functionality.

The Aria library relies heavily on function pointers to take advantage of some object oriented programming concepts like inheritance and polymorphism.

Widgets are typically created by calling their specific “new” function. For instance: “laWidget_New()” will allocate a new basic widget and return a pointer to it (similar to calling new in C++). Calling this function will automatically initialize the widget by calling the constructor for that widget. Deleting widgets is done through the use of the function “laWidget_Delete()”.

Widgets can then be added to layers or other widgets as desired.

Edit Widgets

Edit widgets are a special class of widget that inherits from the EditWidget base implementation instead of Widget. These widgets are capable of becoming the active “edit” widget which means that they will receive any edit events raised by a widget capable of issuing edit events, such as a key pad.

Widget Implementations

The following are descriptions of the widgets offered by Aria:

- Button
- Standard button type widget.
- Can have text and image icon.
- Has a toggle mode.
- Check Box
- Standard check box widget.
- Has built in image for checked and unchecked state.
- Can use custom image for checked and unchecked state.
- Circle
- Widget that draws a circle
- Draw Surface
- Widget that has a callback during its paint loop
- Allows application to make raw HAL draw calls during Aria’s paint loop
- Gradient
- Widget that draws linearly interpolated gradient for its background
- Can use as a parent for other widgets to achieve custom backgrounds
- Group Box

- Widget that functions as a decorated container
- Offers a line border and a horizontally aligned title
- Image
- Widget that draws an image
- Image is clipped to the bounds of the widget.
- Image can be vertically or horizontally aligned to the bounds of the widget
- Image Sequence
- Widget that functions as an automatic image slideshow renderer
- Can add a sequence of widgets and a list of time delays
- Can automatically cycle through list of images without application input
- Key Pad
- A grid of button widgets
- Buttons can be configured to send edit events to the library edit API
- Label
- Widget that draws a string
- Can be aligned vertically and horizontally
- Line
- Widget that draws a line between two specified coordinates
- List
- A list box of strings
- Strings can have icons
- Can be configured to have single, sequential, or multi-selection state
- List Wheel
- A rotating wheel of strings
- Cycles seamlessly through the list
- Responds to drag input
- Panel Widget
- Panels are containers of other widgets, including daughter panels, in support of a parent-child tree of widgets, with the Panel widget as the parent
- Progress Bar
- Widget that fills in a direction based on a given percentage
- Radio Button
- A button that can belong to a group of radio buttons
- Only one button in group can be selected at any one time
- Rectangle
- Widget that draws a rectangle
- Scroll Bar
- A scroll bar that has a configurable scroll range.
- Normally embedded in other widgets like the list box
- Slider
- A widget that slides between a min and max value
- Text Field
- A field of text that can be modified by edit event inputs
- Touch Text
- A widget that draws lines based on input events
- Helps to demonstrate input functionality
- Window
- A container that can be decorated with a title bar
- Title bar can have title text and an icon

Event Management

The Aria state maintains an internal list of events that must get serviced frequently. This is done by called by “laUpdate()”. This is known as the ‘update loop’.

Input Handling

The user interface library has no knowledge of existing hardware but it must provide the means for the user to interact with the scene. Aria thus provides several generic APIs to allow any source to inject input events into the system. These events are stored in the internal event list and are handled during the next update phase.

Scene Rendering

The widget tree is a logical representation of the state of the user interface. The library must be capable of transforming this information into a visual representation that can be sent to the graphics display. The actual rendering is handled by the HAL. The individual widgets contain the algorithms necessary to render themselves but Aria is responsible for telling the widgets when to render themselves. This is known as the ‘paint loop’

The library is responsible for evaluating the widget tree to detect widgets that indicate invalid visual states and managing the redraw. It is essential that widgets only redraw when necessary to avoid needlessly consuming processing resources. It is also important that the library not attempt to draw too much at once as that may starve the rest of the application.

How to Use the Library

```
// initialize the HAL layer
GFX_Initialize();

// initialize the user interface library
laInitialize();

// create a ui context and set active
laContext* uiContext;

iuContext = laContext_Create(0, 0, 0, GFX_COLOR_MODE_RGB_565, NULL);
laContext_SetActive(uiContext);

// create a screen
laScreen* screen;

screen = laScreen_New(LA_FALSE, LA_FALSE, &screenCreate);

// add screen to context
laContext_AddScreen(screen);

// this would be done inside a function called “screenCreate()”
// create layer
laLayer* layer0 = laLayer_New();
laWidget_SetPosition((laWidget*)layer0, 0, 0);
laWidget_SetSize((laWidget*)layer0, 480, 272);

// create a buffer in the layer
laLayer_SetBufferCount(layer0, 1);

// set the layer to the screen
laScreen_SetLayer(screen, 0, layer0);

// create a child widget
laButtonWidget* ButtonWidget1 = laButtonWidget_New();
laWidget_SetPosition((laWidget*)ButtonWidget1, 411, 201);
laWidget_SetSize((laWidget*)ButtonWidget1, 60, 60);
laWidget_SetLocalRedraw((laWidget*)ButtonWidget1, LA_TRUE);
laWidget_SetDrawBackground((laWidget*)ButtonWidget1, LA_FALSE);
laWidget_SetBorderType((laWidget*)ButtonWidget1, LA_WIDGET_BORDER_NONE);
laButtonWidget_SetPressedOffset(ButtonWidget1, 0);
laButtonWidget_SetReleasedEventCallback(ButtonWidget1, &ButtonWidget1_ReleasedEvent);

// add child to parent (layer 0)
laWidget_AddChild((laWidget*)layer0, (laWidget*)ButtonWidget1);

// do this inside application update loop
// update HAL
GFX_Update()

// set ui context as active
laContext_SetActive(uiContext);

// update context (argument is update time in ms)
laUpdate(0);
```

Aria User Interface Library Interface

a) Functions

	Name	Description
	laContext_AddScreen	Add screen to the list of screens in the current context
	laContext_Create	Creates an instance of the Aria user interface library
	laContext_Destroy	Destroys an Aria instance
	laContext_GetActive	Returns the current active context.
	laContext_GetActiveScreen	Returns the active screen of the current context
	laContext_GetActiveScreenIndex	Return the index of the active screen
	laContext_GetColorMode	Returns the color mode of the current context
	laContext_GetDefaultScheme	Returns the pointer to the default scheme of the current context
	laContext_GetEditWidget	Gets the widget that is currently recieving all widget edit events.
	laContext_GetFocusWidget	Return a pointer to the widget in focus
	laContext_GetPremptionLevel	Returns the preemption level for the screen
	laContext_GetScreenRect	Returns the display rectangle structure of the physical display
	laContext_GetStringLanguage	Returns the language index of the current context
	laContext_GetStringTable	Get a pointer to the GFXU_StringTableAsset structure that maintains the strings, associated fonts, etc
	laContext_HideActiveScreen	Hide the active screen
	laContext_RedrawAll	Forces the library to redraw the currently active screen in its entirety.
	laContext_RemoveScreen	Remove the specified screen from the list of screens in the current context
	laContext_SetActive	Make the specified context active
	laContext_SetActiveScreen	Change the active screen to the one specified by the index argument
	laContext_SetActiveScreenChangedCallback	Set the callback function pointer when the screen change event occurs
	laContext_SetEditWidget	Sets the currently active edit widget.
	laContext_SetFocusWidget	Set into focus the widget specified as the argument
	laContext_SetLanguageChangedCallback	Set the callback function pointer when the language change event occurs
	laContext_SetStringLanguage	Set the language index of the current context
	laContext_SetStringTable	Set the StringTable pointer to the specified new StringTableAsset structure
	laContext_Update	Runs the update loop for a library instance.
	laEditWidget_Accept	This is function laEditWidget_Accept.
	laEditWidget_Append	This is function laEditWidget_Append.
	laEditWidget_Backspace	This is function laEditWidget_Backspace.
	laEditWidget_Clear	This is function laEditWidget_Clear.
	laEditWidget_EndEdit	This is function laEditWidget_EndEdit.
	laEditWidget_Set	This is function laEditWidget_Set.
	laEditWidget_StartEdit	This is function laEditWidget_StartEdit.
	laList_Assign	Assignes a new pointer to an index in the list
	laList_Clear	Removes all nodes from a given list
	laList_Copy	Creates a duplicate of an existing list
	laList_Create	Initializes a new linked list
	laList_Destroy	Removes all nodes from a given list and frees the data of each node
	laList_Find	Retrieves the index of a value from the list
	laList_Get	Retrieves a value from the list
	laList_InsertAt	Inserts an item into a list at a given index. All existing from index are shifted right one place.
	laList_PopBack	Removes the last value from the list
	laList_PopFront	Removes the first value from the list
	laList_PushBack	Pushes a new node onto the back of the list
	laList_PushFront	Pushes a new node onto the front of the list

	laList_Remove	Removes an item from the list
	laList_RemoveAt	Removes an item from the list at an index
	laString_Allocate	Attempts to resize the local data buffer for a string.
	laString_Append	Appends a string onto the end of another string
	laString_Capacity	Returns the capacity of a string
	laString_CharAt	Extracts the code point for the character in a string at a given index.
	laString_Clear	Sets a string's length to zero and its string table reference to NULL. Does not free any associated data and preserves capacity.
	laString_Compare	Compares two string objects
	laString_CompareBuffer	Compares a string object and a GFXU_CHAR* buffer
	laString_Copy	Copies the values from one string into another
	laString_CreateFromBuffer	Creates a string object from a GFXU_CHAR buffer and a font asset pointer
	laString_CreateFromCharBuffer	Creates a string object from a const char* buffer and a font asset pointer. This method provides compatibility with standard c-style strings. Input string will be converted from 8-bit with to 32-bit width.
	laString_CreateFromID	Creates a string object that simply references a string in the string table.
	laString_Delete	Deletes all memory associated with a string object
	laString_Destroy	Destroys a string object. This frees the strings internal data buffer, if it exists, sets its string table reference to null, and clears all supporting attributes.
	laString_Draw	Wrapper around GFX Utility string draw function for Aria user interface library. Internal use only.
	laString_ExtractFromTable	Extracts a read-only string from the string table into a modifiable string object. This relies on the active context to indicate which string table to reference as well as which language entry to extract.
	laString_GetCharIndexAtPoint	Given an offset in pixels returns the corresponding character index.
	laString_GetCharOffset	Returns the offset of a given character index in pixels.
	laString_GetCharWidth	Given a character index, gets the width of that character. Only accurate if the string has a font associated with it and that font contains all the characters in the string in question.
	laString_GetHeight	Returns the height of a string by referencing its associated font asset data.
	laString_GetRect	Calculates the rectangle for a given string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup.
	laString_Initialize	Initializes a string struct to default
	laString_Insert	Inserts a string into another string at a given index
	laString_Length	Calculates the length of a string in characters
	laString_New	Allocates a memory for a new string
	laString_ReduceLength	Reduces the length of a string. This simply slides the null terminator to the left and does not affect the string's capacity value.
	laString_Set	Attempts to set the local data buffer of a string to an input buffer
	laString_SetCapacity	Attempts to adjust the capacity of a string
	laString_ToCharBuffer	Extracts the data buffer from a string and copies it into the provided buffer argument.
	laButtonWidget_GetHAlignment	Gets the horizontal alignment setting for a button
	laButtonWidget_GetImageMargin	Gets the distance between the icon and the text
	laButtonWidget_GetImagePosition	Gets the position of the button icon
	laButtonWidget_GetPressed	Gets the pressed state of a button
	laButtonWidget_GetPressedEventCallback	Gets the callback associated with the button pressed event
	laButtonWidget_GetPressedImage	Gets the pressed image asset pointer for a button
	laButtonWidget_GetPressedOffset	Gets the offset of the button internals when pressed
	laButtonWidget_GetReleasedEventCallback	Gets the callback for the button released event
	laButtonWidget_GetReleasedImage	Gets the currently used released icon
	laButtonWidget_GetText	Gets the text for a button. If the button's string has local data then a duplicate of the string will be allocated. The caller is responsible for managing the memory for the duplicated string. If the button string is a string table reference then only the reference ID is copied.

	laButtonWidget_GetToggleable	Gets the value of this button's toggle flag
	laButtonWidget_GetVAlignment	Gets the vertical alignment setting for a button
	laButtonWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laButtonWidget_SetHAlignment	Sets the horizontal alignment value for a button
	laButtonWidget_SetImageMargin	Sets the distance between the icon and text
	laButtonWidget_SetImagePosition	Sets the position of the button icon
	laButtonWidget_SetPressed	Sets the pressed state for a button.
	laButtonWidget_SetPressedEventCallback	Sets the pressed event callback for the button
	laButtonWidget_SetPressedImage	Sets the image to be used as a pressed icon
	laButtonWidget_SetPressedOffset	Sets the offset of the button internals when pressed
	laButtonWidget_SetReleasedEventCallback	Sets the callback for the button released event
	laButtonWidget_SetReleasedImage	Sets the image to be used as the released icon
	laButtonWidget_SetText	Sets the text for a button. If the input string has local data then the data will be copied into the button's local string, causing a memory allocation. If the input string is a string table reference then only the reference will be copied. The input string can be safely modified and the button string will not be affected.
	laButtonWidget_SetToggleable	Enables the toggle mode for a button. When pressed, toggle buttons will stay down until pressed again.
	laButtonWidget_SetVAlignment	Sets the vertical alignment for a button
	laCheckBoxWidget_GetChecked	Gets the checked state of the check box
	laCheckBoxWidget_GetCheckedEventCallback	Gets the checked event callback
	laCheckBoxWidget_GetCheckedImage	Gets the checked image of the check box
	laCheckBoxWidget_GetHAlignment	Gets the horizontal alignment of the check box
	laCheckBoxWidget_GetImageMargin	Gets the distance between the image and the text
	laCheckBoxWidget_GetImagePosition	Gets the image position of the check box
	laCheckBoxWidget_GetText	Gets a copy of the checkbox text. If the text has local data the data will be duplicated. The caller is responsible for managing the memory as appropriate.
	laCheckBoxWidget_GetUncheckedEventCallback	Gets the unchecked event callback
	laCheckBoxWidget_GetUncheckedImage	Gets the unchecked image of the check box
	laCheckBoxWidget_GetVAlignment	Gets the vertical alignment of the check box
	laCheckBoxWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laCheckBoxWidget_SetChecked	Sets the checked state of the check box
	laCheckBoxWidget_SetCheckedEventCallback	Sets the checked event callback
	laCheckBoxWidget_SetCheckedImage	Sets the checked image of the check box
	laCheckBoxWidget_SetHAlignment	Sets the horizontal alignment of the check box.
	laCheckBoxWidget_SetImagePosition	Sets the image position of the check box
	laCheckBoxWidget_SetText	Sets the checkbox text to the input string. If the string has local data the data will be duplicated and copied to the checkboxes internal string.
	laCheckBoxWidget_SetUncheckedEventCallback	Sets the unchecked event callback
	laCheckBoxWidget_SetUncheckedImage	Sets the unchecked image of the check box
	laCheckBoxWidget_SetVAlignment	Sets the vertical alignment of the check box
	laCircleWidget_GetOrigin	Gets the origin coordiates of a circle widget
	laCircleWidget_GetRadius	Gets the radius of a circle widget
	laCircleWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laCircleWidget_SetOrigin	Sets the origin coordiates of a circle widget
	laCircleWidget_SetRadius	Sets the radius of a circle widget
	laDraw_1x2BevelBorder	Internal utility function to draw a 1x2 bevel border
	laDraw_2x1BevelBorder	Internal utility function to draw a 2x1 bevel border
	laDraw_2x2BevelBorder	Internal utility function to draw a 2x2 bevel border
	laDraw_LineBorder	Internal utility function to draw a basic line border

	laDrawSurfaceWidget_GetDrawCallback	Returns the pointer to the currently set draw callback.
	laDrawSurfaceWidget_New	Allocates memory for a new DrawSurface widget.
	laDrawSurfaceWidget_SetDrawCallback	Sets the draw callback pointer for the draw surface widget.
	laEvent_AddEvent	Add the mentioned event callback to the list of events maintained by the current context
	laEvent_ClearList	Clear the event list maintained by the current context.
	laEvent_GetCount	Returns the number of events listed in the current context
	laEvent_ProcessEvents	Processes the screen change as well as touch events
	laEvent_SetFilter	Set callback pointer for current context filter event
	laGradientWidget_GetDirection	Gets the gradient direction value for this widget.
	laGradientWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laGradientWidget_SetDirection	Sets the gradient direction value for this widget.
	laGroupBoxWidget_GetAlignment	Gets the horizontal alignmnet for the group box title text
	laGroupBoxWidget_GetText	Gets the text value for the group box.
	laGroupBoxWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laGroupBoxWidget_SetAlignment	Sets the alignment for the group box title text
	laGroupBoxWidget_SetText	Sets the text value for the group box.
	laImageSequenceWidget_GetImage	Gets the image asset pointer for an entry.
	laImageSequenceWidget_GetImageChangedEventCallback	Gets the image changed event callback pointer.
	laImageSequenceWidget_GetImageCount	Gets the number of image entries for this widget.
	laImageSequenceWidget_GetImageDelay	Gets the image delay for an entry.
	laImageSequenceWidget_GetImageHAlignment	Gets the horizontal alignment for an image entry
	laImageSequenceWidget_GetImageVAlignment	Sets the vertical alignment for an image entry
	laImageSequenceWidget_GetRepeat	Indicates if the widget will repeat through the image entries.
	laImageSequenceWidget_IsPlaying	Indicates if the widget is currently cycling through the image entries.
	laImageSequenceWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laImageSequenceWidget_Play	Starts the widget automatically cycling through the image entries.
	laImageSequenceWidget_Rewind	Resets the current image sequence display index to zero.
	laImageSequenceWidget_SetImage	Sets the image asset pointer for an entry.
	laImageSequenceWidget_SetImageChangedEventCallback	Sets the image changed event callback pointer. This callback is called whenever the active display index is changed.
	laImageSequenceWidget_SetImageCount	Sets the number of image entries for this widget. An image entry that is null will show nothing.
	laImageSequenceWidget_SetImageDelay	Sets the image delay for an entry.
	laImageSequenceWidget_SetImageHAlignment	Sets the horizontal alignment for an image entry.
	laImageSequenceWidget_SetImageVAlignment	Sets the vertical alignment value for an image entry
	laImageSequenceWidget_SetRepeat	Sets the repeat flag for the widget
	laImageSequenceWidget_ShowImage	Sets the active display index to the indicated value.
	laImageSequenceWidget_ShowNextImage	Sets the active display index to the next index value.
	laImageSequenceWidget_ShowPreviousImage	Sets the active display index to the previous index value.
	laImageSequenceWidget_Stop	Stops the widget from automatically cycling through the image entries.
	laImageWidget_GetHAlignment	Gets the image horizontal alignment value.
	laImageWidget_GetImage	Gets the image asset pointer for the widget.
	laImageWidget_GetVAlignment	Gets the image vertical alignment value.
	laImageWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laImageWidget_SetHAlignment	Sets the image horizontal alignment value.
	laImageWidget_SetImage	Sets the image asset pointer for the widget.
	laImageWidget_SetVAlignment	Sets the image vertical alignment value.
	laInput_GetEnabled	Returns the input enabled status of the current context

	laInput_InjectTouchDown	Register and track the touch down event and queue it for handling by associated widgets
	laInput_InjectTouchMoved	Register and track the touch moved event and queue it for handling by associated widgets
	laInput_InjectTouchUp	Register and track the touch up event and queue it for handling by associated widgets
	laInput_SetEnabled	Sets the input status of the current context with the specified input argument
	laKeyPadWidget_GetKeyAction	Gets the key pad cell action for a cell at row/column
	laKeyPadWidget_GetKeyClickEventCallback	Gets the current key click event callback pointer
	laKeyPadWidget_GetKeyDrawBackground	Gets the background type for a key pad cell at row/column
	laKeyPadWidget_GetKeyEnabled	Gets the enabled flag for a cell at a given row/column
	laKeyPadWidget_GetKeyImageMargin	Gets the key pad cell image margin value
	laKeyPadWidget_GetKeyImagePosition	Gets the image position for a key pad cell
	laKeyPadWidget_GetKeyPressedImage	Gets the pressed icon image asset pointer for the display image for a key pad cell
	laKeyPadWidget_GetKeyReleasedImage	Gets the released icon image asset pointer for the display image for a key pad cell
	laKeyPadWidget_GetKeyText	Returns a copy of the display text for a given cell at row/column
	laKeyPadWidget_GetKeyValue	Gets the edit text value for a given key pad cell.
	laKeyPadWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laKeyPadWidget_SetKeyAction	Sets the cell action type for a key pad cell at row/column
	laKeyPadWidget_SetKeyClickEventCallback	Sets the current key click event callback pointer
	laKeyPadWidget_SetKeyEnabled	Sets the enabled flag for a cell at the given row/column
	laKeyPadWidget_SetKeyImageMargin	Sets the key pad cell image margin value for a given cell at row/column
	laKeyPadWidget_SetKeyImagePosition	
	laKeyPadWidget_SetKeyPressedImage	Sets the pressed icon image asset pointer for a key pad cell
	laKeyPadWidget_SetKeyReleasedImage	Sets the released icon image asset pointer for a key pad cell
	laKeyPadWidget_SetKeyText	Sets the display text for a given cell at row/column
	laKeyPadWidget_SetKeyValue	Sets the edit value for a given key pad cell.
	laLabelWidget_GetHAlignment	Gets the text horizontal alignment value.
	laLabelWidget_GetText	Gets the text value for the label.
	laLabelWidget_GetVAlignment	Gets the current vertical text alignment
	laLabelWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laLabelWidget_SetHAlignment	Sets the text horizontal alignment value
	laLabelWidget_SetText	Sets the text value for the label.
	laLabelWidget_SetVAlignment	Sets the vertical text alignment value
	laLayer_Delete	Destructor for the layer object
	laLayer_GetAlphaAmount	Get's the amount of alpha blending for a given layer
	laLayer_GetAlphaEnable	Gets the layer alpha enable flag
	laLayer_GetBufferCount	Return the buffer count for the current layer
	laLayer_GetMaskColor	Returns the mask color value for the current layer
	laLayer_GetMaskEnable	Gets the layer mask enable flag
	laLayer_GetVSync	Gets the layer's vsync flag setting
	laLayer_New	Constructor for a new layer
	laLayer_SetAlphaAmount	Set's the amount of alpha blending for a given layer
	laLayer_SetAlphaEnable	Sets the layer alpha enable flag to the specified value
	laLayer_SetBufferCount	Set the buffer count for the current layer to the specified value
	laLayer_SetMaskColor	Set the mask color value for the current layer to the specified value
	laLayer_SetMaskEnable	Sets the layer mask enable flag to the specified value
	laLayer_SetVSync	Sets the layer's vsync flag.
	laLineWidget_GetEndPoint	Gets the coordinates for the second point of the line.
	laLineWidget_GetStartPoint	Gets the coordinates for the first point of the line.

	laLineWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laLineWidget_SetEndPoint	Sets the cooridnate for the second point of the line
	laLineWidget_SetStartPoint	Sets the cooridnate for the first point of the line
	laListWheelWidget_AppendItem	Appends a new item entry to the list. The initial value of the item will be empty.
	laListWheelWidget_GetAlignment	Gets the horizontal alignment for the list widget
	laListWheelWidget_GetIconMargin	Gets the icon margin value for the list wheel widget
	laListWheelWidget_GetIconPosition	Sets the icon position for the list wheel widget.
	laListWheelWidget_GetItemCount	Gets the number of items currently contained in the list
	laListWheelWidget_GetItemIcon	Gets the pointer to the image asset for the icon for the item at the given index.
	laListWheelWidget_GetItemText	Gets the text value for an item in the list.
	laListWheelWidget_GetSelectedItem	Returns the index of the currently selected item.
	laListWheelWidget_GetSelectedItemChangedEventCallback	Gets the callback for the item selected changed event
	laListWheelWidget_InsertItem	Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.
	laListWheelWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laListWheelWidget_RemoveAllItems	Attempts to remove all items from the list.
	laListWheelWidget_RemoveItem	Attempts to remove an item from the list.
	laListWheelWidget_SelectNextItem	Attempts to move the selected item index to the next item in the list.
	laListWheelWidget_SelectPreviousItem	Attempts to move the selected item index to the previous item in the list.
	laListWheelWidget_SetAlignment	Sets the horizontal alignment mode for the list widget.
	laListWheelWidget_SetIconMargin	Sets the icon margin value for the list widget.
	laListWheelWidget_SetIconPosition	Sets the icon position for the list wheel widget
	laListWheelWidget_SetItemIcon	Sets the icon pointer for a given index.
	laListWheelWidget_SetItemText	Sets the text value for an item in the list.
	laListWheelWidget_SetSelectedItem	Attempts to set the selectedi tem index
	laListWheelWidget_SetSelectedItemChangedEventCallback	
	laListWidget_AppendItem	Appends a new item entry to the list. The initial value of the item will be empty.
	laListWidget_DeselectAll	Attempts to set all item states as not selected.
	laListWidget_GetAlignment	Gets the horizontal alignment for the list widget
	laListWidget_GetAllowEmptySelection	Returns true of the list allows an empty selection set
	laListWidget_GetFirstSelectedItem	Returns the lowest selected item index.
	laListWidget_GetIconMargin	Gets the icon margin value for the list widget
	laListWidget_GetIconPosition	Gets the icon position for the list
	laListWidget_GetItemCount	Gets the number of items currently contained in the list
	laListWidget_GetItemIcon	Gets the pointer to the image asset for the icon for the item at the given index.
	laListWidget_GetItemSelected	Returns true if the item at the given index is currently selected.
	laListWidget_GetItemText	Gets the text value for an item in the list.
	laListWidget_GetLastSelectedItem	Returns the highest selected item index.
	laListWidget_GetSelectedItemChangedEventCallback	Gets the callback for the item selected changed event
	laListWidget_GetSelectionCount	Returns the number of selected items in the list.
	laListWidget_GetSelectionMode	Gets the selection mode for the list
	laListWidget_InsertItem	Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.
	laListWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laListWidget_RemoveAllItems	Attempts to remove all items from the list.
	laListWidget_RemoveItem	Attempts to remove an item from the list.
	laListWidget_SelectAll	Attempts to set all item states to selected.

	laListWidget_SetAlignment	Sets the horizontal alignment mode for the list widget.
	laListWidget_SetAllowEmptySelection	Configures the list to allow an empty selection set.
	laListWidget_SetIconMargin	Sets the icon margin value for the list widget.
	laListWidget_SetIconPosition	Sets the icon position for the list widget
	laListWidget_SetItemIcon	Sets the icon pointer for a given index.
	laListWidget_SetItemSelected	Attempts to set the item at idx as selected.
	laListWidget_SetItemText	Sets the text value for an item in the list.
	laListWidget_SetItemVisible	
	laListWidget_SetSelectedItemChangedEventCallback	Sets the callback for the item selected changed event
	laListWidget_SetSelectionMode	Set the list selection mode
	laListWidget_ToggleItemSelected	Attempts to toggle the selected state of the item at idx.
	laProgressBarWidget_GetDirection	Gets the fill direction value for a progress bar widget
	laProgressBarWidget_GetValue	Gets the percentage value for a progress bar.
	laProgressBarWidget_GetValueChangedEventCallback	Gets the currently set value changed event callback.
	laProgressBarWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laProgressBarWidget_SetDirection	Sets the fill direction for a progress bar widget
	laProgressBarWidget_SetValue	Sets the percentage value for a progress bar. Valid values are 0 - 100.
	laProgressBarWidget_SetValueChangedCallback	Sets the desired value changed event callback pointer
	laRadioButtonGroup_AddButton	Add a button widget to the button list of the selected Radio button group.
	laRadioButtonGroup_Create	This function creates a GFX_GOL_RADIOBUTTON group with the provided button list.
	laRadioButtonGroup_Destroy	This function destroys the GFX_GOL_RADIOBUTTON group
	laRadioButtonGroup_RemoveButton	Remove a button widget to the button list of the selected Radio button group.
	laRadioButtonGroup_SelectButton	Select the button widget specified from the button list for the Radio button group.
	laRadioButtonWidget_GetDeselectedEventCallback	Gets the current radio button deselected event callback
	laRadioButtonWidget_GetGroup	Returns the pointer to the currently set radio button group.
	laRadioButtonWidget_GetHAlignment	Gets the horizontal alignment setting for a button
	laRadioButtonWidget_GetImageMargin	Gets the distance between the icon and the text
	laRadioButtonWidget_GetImagePosition	Gets the current image position setting for the radio button
	laRadioButtonWidget_GetSelected	Returns true if this radio button is currently selected
	laRadioButtonWidget_GetSelectedEventCallback	Gets the current radio button selected event callback
	laRadioButtonWidget_GetSelectedImage	Gets the selected image asset pointer for a button
	laRadioButtonWidget_GetText	Gets the text value for the button.
	laRadioButtonWidget_GetUnselectedImage	Gets the image asset pointer currently used as the unselected icon
	laRadioButtonWidget_GetVAlignment	Sets the vertical alignment for a button
	laRadioButtonWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laRadioButtonWidget_SetDeselectedEventCallback	Sets the deselected callback pointer
	laRadioButtonWidget_SetHAlignment	Sets the horizontal alignment value for a button
	laRadioButtonWidget_SetImagePosition	Sets the image relative position setting for the radio button
	laRadioButtonWidget_SetSelected	Sets this button as selected.
	laRadioButtonWidget_SetSelectedEventCallback	Sets the radio button selected event callback
	laRadioButtonWidget_SetSelectedImage	Sets the image to be used as a selected icon
	laRadioButtonWidget_SetText	Sets the text value for the button.
	laRadioButtonWidget_SetUnselectedImage	Sets the asset pointer for the radio button's unselected image icon
	laRadioButtonWidget_SetVAlignment	Sets the vertical alignment for a button
	laRectangleWidget_GetThickness	Gets the rectangle border thickness setting
	laRectangleWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

	laScheme_Initialize	Initialize the scheme to the default values as per the specified color mode.
	laScreen_Delete	Frees all memory for all layers and widgets for this screen
	laScreen_GetHideEventCallback	Returns the hide call back event function pointer for the specified screen
	laScreen_GetLayerIndex	Returns the index of the layer for the screen specified.
	laScreen_GetOrientation	Returns the orientation object associated with the specified screen
	laScreen_GetShowEventCallback	Returns the show call back event function pointer for the specified screen
	laScreen_Hide	Hide the currently active screen This function has been deprecated in favor of laContext_SetActiveScreen
	laScreen_New	Create a new screen, initialize it to the values specified.
	laScreen_SetHideEventCallback	Set the hide call back event function pointer for the specified screen
	laScreen_SetLayer	Assigns the provided layer pointer to the screen at the given index This function has been deprecated in favor of laContext_SetActiveScreen
	laScreen_SetOrientation	Sets the orientation object to the specified screen
	laScreen_SetShowEventCallback	Set the show call back event function pointer for the specified screen
	laScreen_Show	Make the specified screen active and show it on the display
	laScrollBarWidget_GetExtentValue	Gets the current scroll bar extent value
	laScrollBarWidget_GetMaxinumValue	Gets the maximum scroll value
	laScrollBarWidget_GetOrientation	Gets the orientation value for the scroll bar
	laScrollBarWidget_GetScrollPercentage	Gets the current scroll value as a percentage
	laScrollBarWidget_GetScrollValue	Gets the current scroll value
	laScrollBarWidget_GetStepSize	Gets the current discreet step size
	laScrollBarWidget_GetValueChangedEventCallback	Gets the current value changed callback function pointer
	laScrollBarWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laScrollBarWidget_SetExtentValue	Sets the scroll bar extent value
	laScrollBarWidget_SetMaximumValue	Sets the maximum scroll value
	laScrollBarWidget_SetOrientation	Sets the orientation value of the scroll bar
	laScrollBarWidget_SetScrollPercentage	Sets the current scroll value using a percentage. Percentage should be a value from 0 - 100
	laScrollBarWidget_SetScrollValue	Sets the current scroll value
	laScrollBarWidget_SetStepSize	Sets the current step size
	laScrollBarWidget_SetValueChangedEventCallback	Sets the value changed event callback pointer
	laScrollBarWidget_StepBackward	Moves the scroll value back by the current step size
	laScrollBarWidget_StepForward	Moves the scroll value forward by the current step size
	laSliderWidget_GetGripSize	Gets the current grip size of the slider
	laSliderWidget_GetMaxinumValue	Gets the maximum value for the slider
	laSliderWidget_GetMininumValue	Gets the minimum value for the slider
	laSliderWidget_GetOrientation	Gets the orientation value for the slider
	laSliderWidget_GetSliderPercentage	Gets the slider value as a percentage
	laSliderWidget_GetSliderValue	Gets the current slider value
	laSliderWidget_GetValueChangedEventCallback	Gets the current value changed event callback pointer
	laSliderWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laSliderWidget_SetGripSize	Sets the grip size of the slider
	laSliderWidget_SetMaximumValue	Sets the maximum value for the slider
	laSliderWidget_SetMinimumValue	Sets the minimum value for the slider
	laSliderWidget_SetOrientation	
	laSliderWidget_SetSliderPercentage	Sets the slider value using a percentage. Value must be from 0 - 100.
	laSliderWidget_SetSliderValue	Sets the current slider value
	laSliderWidget_SetValueChangedEventCallback	Sets the value changed event callback pointer
	laSliderWidget_Step	Moves the slider by a given amount

	laTextFieldWidget_GetAlignment	Gets the text horizontal alignment value.
	laTextFieldWidget_GetCursorDelay	Gets the current cursor delay.
	laTextFieldWidget_GetCursorEnabled	Gets the cursor enabled value
	laTextFieldWidget_GetCursorPosition	Gets the current edit cursor position
	laTextFieldWidget_GetText	Gets the text value for the box.
	laTextFieldWidget_GetTextChangedEventCallback	Gets the current text changed event callback pointer
	laTextFieldWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laTextFieldWidget_SetAlignment	Sets the text horizontal alignment value
	laTextFieldWidget_SetCursorDelay	Sets the cursor delay value
	laTextFieldWidget_SetCursorEnabled	Sets the cursor enabled value flag
	laTextFieldWidget_SetCursorPosition	Sets the position of the cursor
	laTextFieldWidget_SetText	Sets the text value for the box.
	laTextFieldWidget_SetTextChangedEventCallback	Sets the text changed event callback pointer
	laTouchTest_AddPoint	Adds a point to the touch test widget. The point will then be displayed.
	laTouchTest_ClearPoints	Clears all of the existing touch points
	laTouchTestWidget_GetPointAddedEventCallback	Gets the current point added event callback
	laTouchTestWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laTouchTestWidget_SetPointAddedEventCallback	Sets the point added event callback
	laUtils_ArrangeRectangle	Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the image rectangle.
	laUtils_ArrangeRectangleRelative	Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the text rectangle.
	laUtils_ChildIntersectsParent	Performs an intersection test between a parent widget and a child widget
	laUtils_ClipRectToParent	Clips a rectangle to the parent of a widget
	laUtils_GetLayer	Finds the root parent of a widget, which should be a layer
	laUtils_ListOcclusionCullTest	Performs an occlusion test on a list of widgets an a rectangular area. This attempts to find only the topmost widgets for the given area. If a widget is completely occluded then it is removed from the list. Any widgets that remain in the list should be redrawn by the rasterizer.
	laUtils_OcclusionCullTest	Performs an occlusion test for a widget in the tree. A widget is occluded if it is completely covered by one or more widgets. This is useful for culling widgets before the rasterizing phase.
	laUtils_Pick	Finds the top-most visible widget in the tree at the given coordinates.
	laUtils_PickRect	Finds all of the visible widgets in the given rectangular area.
	laUtils_PointScreenToLocalSpace	Converts a point from layer space into the local space of a widget
	laUtils_RectFromParentSpace	Converts a rectangle from widget parent space to widget local space
	laUtils_RectToLayerSpace	Converts a rectangle from widget local space to layer space
	laUtils_RectToParentSpace	Converts a rectangle from widget local space to widget parent space. Widget must be a child of a layer for this to function.
	laUtils_RectToScreenSpace	Converts a rectangle from widget local space to screen space
	laWidget_AddChild	Adds the child to the parent widget specified in the argument
	laWidget_Delete	Delete the widget object specified
	laWidget_GetAlphaAmount	Return the widget's global alpha amount
	laWidget_GetAlphaEnable	Return the alpha enable property of the widget
	laWidget_GetBorderType	Return the border type associated with the widget object
	laWidget_GetChildAtIndex	Fetches the child at the specified index from the children list of the specified parent widget
	laWidget_GetChildCount	Returns the size of the children list of the specified parent widget
	laWidget_GetCumulativeAlphaAmount	Calculates the cumulative alpha amount for a hierarchy of widgets

	laWidget_GetCumulativeAlphaEnable	Determines if this or any ancestor widget has alpha enabled
	laWidget_GetEnabled	Returns the boolean value of the widget enabled property
	laWidget_GetHeight	Returns the widget rectangles height
	laWidget_GetIndexOfChild	Fetches the index of the child from the children list of the specified parent widget
	laWidget_GetMargin	Returns the margin value associated with the widget in the laMargin pointer
	laWidget_GetScheme	Returns the scheme associated with the specified widget
	laWidget_GetVisible	Returns the boolean value of the widget visible property
	laWidget_GetWidth	Returns the widget rectangles width
	laWidget_GetX	Returns the widget rectangles upper left corner x-coordinate
	laWidget_GetY	Returns the widget rectangles upper left corner y-coordinate
	laWidget_HasFocus	Checks if the widget specified has focus in the current context
	laWidget_Invalidate	Invalidates the specified widget.
	laWidget_isOpaque	Returns true if the widget is considered opaque.
	laWidget_New	Create a new widget.
	laWidget_OverrideTouchDownEvent	Replace the TouchDownEvent callback for the widget with the new function pointer specified
	laWidget_OverrideTouchMovedEvent	Replace the TouchMovedEvent callback for the widget with the new function pointer specified
	laWidget_OverrideTouchUpEvent	Replace the TouchUpEvent callback for the widget with the new function pointer specified
	laWidget_RectToLayerSpace	Transforms a widget rectangle from local space to its root layer space.
	laWidget_RectToParentSpace	Returns the rectangle containing the parent of the widget specified
	laWidget_RectToScreenSpace	Transforms a widget rectangle from local space to screen space coordinates.
	laWidget_RemoveChild	Removes the child from the parent widget specified in the argument
	laWidget_Resize	Changes the widget size by the new defined width and height increments.
	laWidget_SetAlphaAmount	Set the widget's global alpha amount to the specified alpha amount
	laWidget_SetAlphaEnable	Set the alpha enable property of the widget with the boolean value specified
	laWidget_SetBorderType	Set the border type associated with the widget object
	laWidget_SetEnabled	Sets the boolean value of the widget enabled property
	laWidget_SetFocus	Set the widget into focus for the current context.
	laWidget_SetHeight	Sets the widget's height value
	laWidget_SetMargins	Set the margin value for left, right, top and bottom margins associated with the widget
	laWidget_SetParent	Sets the parent of the child widget to that specified in the argument list
	laWidget_SetPosition	Changes the widget position to the new defined x and y coordinates.
	laWidget_SetScheme	Sets the scheme variable for the specified widget
	laWidget_SetSize	Changes the widget size to the new defined width and height dimensions.
	laWidget_SetVisible	Sets the boolean value of the widget visible property
	laWidget_SetWidth	Sets the widget's width value
	laWidget_SetX	Sets the widget's x coordinate position
	laWidget_SetY	Sets the widget's y coordinate position
	laWidget_Translate	Changes the widget position by moving the widget by the defined x and y increments.
	laWindowWidget_GetIcon	Gets the currently used window icon
	laWindowWidget_GetIconMargin	Gets the current image icon margin
	laWindowWidget_GetTitle	Gets the title text for this window.
	laWindowWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laWindowWidget_SetIcon	Sets the image to be used as a window icon
	laWindowWidget_SetIconMargin	Sets the image icon margin

	laWindowWidget_SetTitle	Sets the title text for the window.
	laCheckBoxWidget_SetImageMargin	Sets the distance between the image and the text
	laContext_SetPreemptionLevel	Set the preemption level to the specified value
	laKeyPadWidget_SetKeyBackgroundType	Sets the background type for a key pad cell at row/column
	laLayer_GetAllowInputPassThrough	Gets the layer's input passthrough setting
	laLayer_GetEnabled	Returns the boolean value of the layer enabled property
	laLayer_SetAllowInputPassthrough	Sets the layer's input passthrough flag.
	laLayer_SetEnabled	Sets the boolean value of the layer enabled property
	laListWheelWidget_GetFlickInitSpeed	Returns the flick init speed for the wheel.
	laListWheelWidget_GetIndicatorArea	Returns the spacing for the selected item indicator bars.
	laListWheelWidget_GetMaxMomentum	Returns the maximum momentum value for the wheel.
	laListWheelWidget_GetMomentumFalloffRate	Returns the momentum falloff rate for the wheel.
	laListWheelWidget_GetRotationUpdateRate	Returns the wheel rotation update rate.
	laListWheelWidget_GetShaded	Returns true if the list is using gradient shading to illustrate depth
	laListWheelWidget_GetShowIndicators	Returns true if the list is displaying its selected item indicators
	laListWheelWidget_GetVisibleItemCount	Returns the list's visible item count
	laListWheelWidget_SetFlickInitSpeed	Configures the flick init speed for the list wheel
	laListWheelWidget_SetIndicatorArea	Configures the display area for the list selection indicator bars
	laListWheelWidget_SetMaxMomentum	Configures the maximum momentum value for the wheel
	laListWheelWidget_SetMomentumFalloffRate	Configures the momentum falloff rate for the wheel
	laListWheelWidget_SetRotationUpdateRate	Configures the rotation update rate for a wheel
	laListWheelWidget_SetShaded	Configures the list to use gradient or flat background shading
	laListWheelWidget_SetShowIndicators	Configures the list to display the selected item indicator bars
	laListWheelWidget_SetVisibleItemCount	Sets the number of visible items in the list. Must be greater than or equal to three and must be an odd number.
	laRadioButtonWidget_SetImageMargin	Sets the distance between the icon and text
	laScreen_GetMirrored	Returns the mirror setting for the specified screen
	laScreen_SetMirrored	Sets the mirror setting for the specified screen
	laString_Remove	Removes a number of characters from a string at a given index
	laTextFieldWidget_SetClearOnFirstEdit	Sets the flag to indicate that the text field will be cleared on first edit.
	laUtils_PickFromLayer	Finds the top-most visible widget in a layer at the given coordinates.
	laUtils_PointToLayerSpace	Converts a point from widget space into layer space
	laUtils_ScreenToMirroredSpace	Takes a point in screen space and returns a transformed version in mirrored space.
	laUtils_ScreenToOrientedSpace	Takes a point in screen space and returns a transformed version in oriented space.
	laUtils_WidgetLocalRect	Returns the bounding rectangle of a widget in local space
	laWidget_GetBackgroundType	Return the property value 'background type' associated with the widget object
	laWidget_GetOptimizationFlags	Returns the optimization flags for the widget
	laWidget_SetBackgroundType	Set the property value 'background type' associated with the widget object
	laWidget_SetOptimizationFlags	Sets the optimizations for a widget
	laRectangleWidget_SetThickness	Sets the rectangle border thickness setting
	laString_DrawClipped	Wrapper around GFX Utility string draw function for Aria user interface library. Draws only a clipped area of a string. Internal use only.
	laString_IsEmpty	Returns a boolean indicating if the provided string contains data or has a link to the string table.
	laUtils_GetNextHighestWidget	Gets the next highest Z order widget in the tree from 'wgt'
	laUtils_RectFromLayerSpace	Converts a rectangle from layer space to widget local space
	laUtils_WidgetIsOccluded	This is function laUtils_WidgetIsOccluded.
	laUtils_WidgetLayerRect	Returns the bounding rectangle of a widget in layer space
	laWindowWidget_GetIconRect	This is function laWindowWidget_GetIconRect.
	laWindowWidget_GetTextRect	This is function laWindowWidget_GetTextRect.
	laWindowWidget_GetTitleBarRect	internal use only

	laContext_IsDrawing	Indicates if any layers of the active screen are currently drawing a frame.
	laContext_IsLayerDrawing	Indicates if the layer at the given index of the active screen is currently drawing.
	laLayer_IsDrawing	Queries a layer to find out if it is currently drawing a frame.
	laLayer_GetInputRect	Gets the layer's input rectangle.
	laLayer_GetInputRectLocked	Gets the layer's input rect locked flag
	laLayer_SetInputRect	Sets the layer's input rect dimensions.
	laLayer_SetInputRectLocked	Sets the layer's input rect locked flag.
	laScreen_GetLayerSwapSync	Returns the layer swap sync setting for the specified screen
	laScreen_SetLayerSwapSync	Sets the layer swap sync setting for the specified screen
	laWidget_DeleteAllDescendants	Deletes all of the descendants of the given widget.
	laImageWidget_SetCallBackEnd	This is function laImageWidget_SetCallBackEnd.
	laImageWidget_SetCallBackStart	This is function laImageWidget_SetCallBackStart.
	laString_DrawSubStringClipped	Wrapper around GFX Utility string draw function for Aria user interface library. Draws the substring between the start and end offset, and draws only the section of the string within the clipping rectangle. Internal use only.
	laString_GetLineRect	Calculates the rectangle for a line in a string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The line ends when a line feed or end of string is reached.
	laString_GetMultiLineRect	Calculates the rectangle for a given multi-line string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The height the sum of the heights of the bounding rectangles for each line and the width is the widest among the bounding rectangles.

b) Data Types and Constants

	Name	Description
	laBool_t	libaria bool values
	laContext_t	An instance of the Aria user interface library.
	laEditWidget_t	Specifies the edit widget structure to manage all properties and events associated with edit widgets
	laList_t	Linked list definition
	laListNode_t	Linked list node definition
	laMargin_t	libaria margin values
	laPreemptionLevel	libaria pre-emption level values
	laRelativePosition	libaria relative position values
	laResult_t	libaria results (success and failure codes).
	laString_t	String definition
	GFXU_StringTableAsset	Describes a string table asset. There is typically only ever one of these defined at any one time. header - standard asset header languageCount - the number of languages in the string table stringCount - the number of strings in the string table stringIndexTable - the pointer to the string index table. the string index table is a table that contains all of the unique strings defined in the string table. fontTable - the font table contains an array of pointers to all defined font assets that the string table references fontIndexTable - the font index table is a table... more
	laBool	libaria bool values
	laContext	An instance of the Aria user interface library.
	laContext_ActiveScreenChangedCallback_FnPtr	Callback pointer for the active screen change notification.
	laContext_LanguageChangedCallback_FnPtr	Callback pointer for when the language change event occurs.
	laEditWidget	Specifies the edit widget structure to manage all properties and events associated with edit widgets
	laEditWidget_Accept_FnPtr	This is type laEditWidget_Accept_FnPtr.
	laEditWidget_Append_FnPtr	This is type laEditWidget_Append_FnPtr.
	laEditWidget_Backspace_FnPtr	This is type laEditWidget_Backspace_FnPtr.
	laEditWidget_Clear_FnPtr	This is type laEditWidget_Clear_FnPtr.
	laEditWidget_EndEdit_FnPtr	This is type laEditWidget_EndEdit_FnPtr.

	laEditWidget_Set_FnPtr	This is type laEditWidget_Set_FnPtr.
	laEditWidget_StartEdit_FnPtr	This is type laEditWidget_StartEdit_FnPtr.
	laHAlignment	libaria horizontal alignment values
	laList	Linked list definition
	laListNode	Linked list node definition
	laMargin	libaria margin values
	laResult	libaria results (success and failure codes).
	laScreen	The structure to maintain the screen related variables and event handling
	laString	String definition
	laVAlignment	libaria vertical alignment values
	laBorderType_t	Specifies the different border types used for the widgets in the library
	laButtonState_t	Controls the button pressed state
	laButtonWidget_t	Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.
	laCheckBoxWidget_t	Implementation of a checkbox widget.
	laCircleWidget_t	Implementation of a circle widget.
	laDrawSurfaceWidget_t	Implementation of a Drawsurface widget.
	laEvent_t	Basic UI event definition
	laEventID_t	Defines internal event type IDs
	laEventState_t	Structure to manage the event lists, state and call back pointers
	laGestureID_t	Placeholder for eventual gesture support.
	laGradientWidget_t	Gradient widget struct definition.
	laGradientWidgetDirection_t	Implementation of a gradient widget.
	laGroupBoxWidget_t	Group box struct definition.
	laImageSequenceEntry_t	Image sequence entry definition
	laImageSequenceWidget_t	Image sequence widget struct definition
	laImageWidget_t	Image widget struct definition
	laInput_TouchDownEvent_t	Register and handle the touch press detect event
	laInput_TouchMovedEvent_t	Register and handle the touch coordinates changed event
	laInput_TouchUpEvent_t	Register and handle the touch release detect event
	laInputState_t	Maintain a history of touch states; currently libaria keeps track of the last touch state only.
	laKey_t	All values possible for key entry from the libaria keyboard widget
	laKeyPadCell_t	Defines a key pad cell struct
	laKeyPadCellAction_t	Defines an assigned action to a key pad cell
	laKeyPadWidget_t	Defines a key pad widget struct
	laLabelWidget_t	Implementation of a label widget struct
	laLayer_t	Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.
	laLayerBuffer_t	Structure to maintain the buffer type and track the buffer location for each layer
	laLayerBufferType_t	Defines the type of a layer. If the layer has an explicit address then Aria tries to set that through the HAL when the layer is being set up.
	laLineWidget_t	Defines the implementation of a line widget struct
	laListItem_t	Defines a list item struct
	laListWheelItem_t	Implementation of a list wheel widget item struct
	laListWheelWidget_t	Implementation of a list wheel widget struct
	laListWidget_SelectionMode_t	Defines the list selection modes
	laListWidget_t	Defines the implementation of a list widget
	laMouseButton_t	All values possible for mouse key entry from the libaria mouse input
	laProgressBarDirection_t	Defines the valid values for the progress bar widget fill directions.
	laProgressBarWidget_t	Implementation of a progressbar widget struct
	laRadioButtonGroup_t	Defines the structure used for the Radio Button group.
	laRadioButtonWidget_t	Implementation of a radio button widget struct
	laRectangleWidget_t	Implementation of a rectangle widget struct
	laScheme_t	This structure specifies the style scheme components of an object.
	laScreen_t	The structure to maintain the screen related variables and event handling

	laScreenOrientation_t	Possible values for screen orientation.
	laScrollBarOrientation_t	Defines the scroll bar direction values
	laScrollBarState_t	Defines the various scroll bar state values
	laScrollBarWidget_t	Implementation of a scroll bar widget.
	laSliderOrientation_t	Slider orientations
	laSliderState_t	Describes various slider states
	laSliderWidget_t	Implementation of a slider widget struct
	laTextFieldWidget_t	Implementation of a text field widget.
	laTouchState_t	Manage the touch input state and track the touch coordinate
	laTouchTestState_t	Touch test states
	laTouchTestWidget_t	Implementation of a touch test widget struct
	laWidget_t	Specifies Graphics widget structure to manage all properties and events associated with the widget
	laWidgetDirtyState_t	Specifies the different dirty states the widget can be assigned
	laWidgetDrawState_t	Specifies the different draw states the widget can be assigned
	laWidgetEvent_t	Basic widget event definition
	laWidgetType_t	Specifies the different widget types used in the library
	laWindowWidget_t	Implementation of a window widget struct
	GFX_Point	A two dimensional Cartesian point.
	GFX_Rect	A rectangle definition.
	laBorderType	Specifies the different border types used for the widgets in the library
	laButtonState	Controls the button pressed state
	laButtonWidget	Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.
	laButtonWidget_PressedEvent	This is type laButtonWidget_PressedEvent.
	laButtonWidget_ReleasedEvent	This is type laButtonWidget_ReleasedEvent.
	laCheckBoxWidget	Implementation of a checkbox widget.
	laCheckBoxWidget_CheckedEvent	This is type laCheckBoxWidget_CheckedEvent.
	laCheckBoxWidget_UncheckedEvent	This is type laCheckBoxWidget_UncheckedEvent.
	laCircleWidget	Implementation of a circle widget.
	laDrawSurfaceWidget	Implementation of a Drawsurface widget.
	laDrawSurfaceWidget_DrawCallback	This is type laDrawSurfaceWidget_DrawCallback.
	laEvent	Basic UI event definition
	laEvent_FilterEvent	Function pointer to define an event filter. Event filters allow a receiver to discard undesirable events
	laEventID	Defines internal event type IDs
	laEventState	Structure to manage the event lists, state and call back pointers
	laGestureID	Placeholder for eventual gesture support.
	laGradientWidget	Gradient widget struct definition.
	laGradientWidgetDirection	Implementation of a gradient widget.
	laGroupBoxWidget	Group box struct definition.
	laImageSequenceEntry	Image sequence entry definition
	laImageSequenceImageChangedEvent_FnPtr	This is type laImageSequenceImageChangedEvent_FnPtr.
	laImageSequenceWidget	Image sequence widget struct definition
	laImageWidget	Image widget struct definition
	laInput_TouchDownEvent	Register and handle the touch press detect event
	laInput_TouchMovedEvent	Register and handle the touch coordinates changed event
	laInput_TouchUpEvent	Register and handle the touch release detect event
	laInputState	Maintain a history of touch states; currently libaria keeps track of the last touch state only.
	laKey	All values possible for key entry from the libaria keyboard widget
	laKeyPadCell	Defines a key pad cell struct
	laKeyPadCellAction	Defines an assigned action to a key pad cell
	laKeyPadWidget	Defines a key pad widget struct
	laKeyPadWidget_KeyClickEvent	This is type laKeyPadWidget_KeyClickEvent.
	laLabelWidget	Implementation of a label widget struct

	laLayer	Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.
	laLayerBuffer	Structure to maintain the buffer type and track the buffer location for each layer
	laLayerBufferType	Defines the type of a layer. If the layer has an explicit address then Aria tries to set that through the HAL when the layer is being set up.
	laLineWidget	Defines the implementation of a line widget struct
	laListItem	Defines a list item struct
	laListWheelItem	Implementation of a list wheel widget item struct
	laListWheelWidget	Implementation of a list wheel widget struct
	laListWheelWidget_SelectedItemChangedEvent	This is type laListWheelWidget_SelectedItemChangedEvent.
	laListWidget	Defines the implementation of a list widget
	laListWidget_ItemSelectedChangedEvent	This is type laListWidget_ItemSelectedChangedEvent.
	laListWidget_SelectedItemChangedEvent	This is type laListWidget_SelectedItemChangedEvent.
	laListWidget_SelectionMode	Defines the list selection modes
	laMouseButton	All values possible for mouse key entry from the libaria mouse input
	laProgressBar	This is type laProgressBar.
	laProgressBar_ValueChangedEventCallback	This is type laProgressBar_ValueChangedEventCallback.
	laProgressBarDirection	Defines the valid values for the progress bar widget fill directions.
	laProgressBarWidget	Implementation of a progressbar widget struct
	laRadioButtonGroup	Defines the structure used for the Radio Button group.
	laRadioButtonWidget	Implementation of a radio button widget struct
	laRadioButtonWidget_DeselectedEvent	This is type laRadioButtonWidget_DeselectedEvent.
	laRadioButtonWidget_SelectedEvent	This is type laRadioButtonWidget_SelectedEvent.
	laRectangleWidget	Implementation of a rectangle widget struct
	laScheme	This structure specifies the style scheme components of an object.
	laScreen_CreateCallback_FnPtr	Callback pointer for a new screen create event notification. This is called when the library attempts to create a screen.
	laScreen_ShowHideCallback_FnPtr	Callback pointer for the active screen show or hide event change notification.
	laScreenOrientation	Possible values for screen orientation.
	laScrollBarOrientation	Defines the scroll bar direction values
	laScrollBarState	Defines the various scroll bar state values
	laScrollBarWidget	Implementation of a scroll bar widget.
	laScrollBarWidget_ValueChangedEvent	This is type laScrollBarWidget_ValueChangedEvent.
	laSliderOrientation	Slider orientations
	laSliderState	Describes various slider states
	laSliderWidget	Implementation of a slider widget struct
	laSliderWidget_ValueChangedEvent	This is type laSliderWidget_ValueChangedEvent.
	laTextFieldWidget	Implementation of a text field widget.
	laTextFieldWidget_TextChangedCallback	This is type laTextFieldWidget_TextChangedCallback.
	laTouchState	Manage the touch input state and track the touch coordinate
	laTouchTestState	Touch test states
	laTouchTestWidget	Implementation of a touch test widget struct
	laTouchTestWidget_PointAddedEventCallback	This is type laTouchTestWidget_PointAddedEventCallback.
	laWidget	Specifies Graphics widget structure to manage all properties and events associated with the widget
	laWidget_Constructor_FnPtr	This is type laWidget_Constructor_FnPtr.
	laWidget_Destructor_FnPtr	This is type laWidget_Destructor_FnPtr.
	laWidget_DrawFunction_FnPtr	This is type laWidget_DrawFunction_FnPtr.
	laWidget_Focus_FnPtr	This is type laWidget_Focus_FnPtr.
	laWidget_Moved_FnPtr	This is type laWidget_Moved_FnPtr.
	laWidget_Paint_FnPtr	This is type laWidget_Paint_FnPtr.
	laWidget_Resized_FnPtr	This is type laWidget_Resized_FnPtr.
	laWidget_TouchDownEvent_FnPtr	This is type laWidget_TouchDownEvent_FnPtr.
	laWidget_TouchMovedEvent_FnPtr	This is type laWidget_TouchMovedEvent_FnPtr.
	laWidget_TouchUpEvent_FnPtr	This is type laWidget_TouchUpEvent_FnPtr.
	laWidget_Update_FnPtr	This is type laWidget_Update_FnPtr.

	laWidgetDirtyState	Specifies the different dirty states the widget can be assigned
	laWidgetDrawState	Specifies the different draw states the widget can be assigned
	laWidgetEvent	Basic widget event definition
	laWidgetType	Specifies the different widget types used in the library
	laWindowWidget	Implementation of a window widget struct
	laBackgroundType_t	Specifies the different background types used for the widgets in the library
	laWidgetOptimizationFlags_t	Specifies the different draw optimization flags for a widget
	laBackgroundType	Specifies the different background types used for the widgets in the library
	laWidget_LanguageChangingEvent_FnPtr	This is type laWidget_LanguageChangingEvent_FnPtr.
	laWidgetOptimizationFlags	Specifies the different draw optimization flags for a widget
	LA_DEFAULT_SCHEME_COLOR_MODE	This is macro LA_DEFAULT_SCHEME_COLOR_MODE.
	LA_STRING_NULLIDX	This is macro LA_STRING_NULLIDX.
	DEFAULT_BORDER_MARGIN	This is macro DEFAULT_BORDER_MARGIN.
	LA_IMAGESEQ_RESTART	This is macro LA_IMAGESEQ_RESTART.
	LA_INPUT_PRIMARY_ID	This is macro LA_INPUT_PRIMARY_ID.
	LA_MAX_TOUCH_STATES	This is macro LA_MAX_TOUCH_STATES.
	LA_TOUCHTEST_MEMORY_SIZE	This is macro LA_TOUCHTEST_MEMORY_SIZE.
	NUM_BUTTONS	This is macro NUM_BUTTONS.
	NUM_KEYS	This is macro NUM_KEYS.
	laLayer_AddDamageRect	Adds a damaged rectangle to the list. Damage rectangles are used in minimal redraw algorithms.
	laEventResult	Defines what happened when processing an event
	laLayerFrameState	Defines the frame state of a layer. Certain actions must only be performed at the start of a new frame and other actions must wait until the end of the current frame.
	laRectArray	This is type laRectArray.
	laWidget_InvalidateBorderAreas_FnPtr	This is type laWidget_InvalidateBorderAreas_FnPtr.
	laEventResult_t	Defines what happened when processing an event
	laLayerFrameState_t	Defines the frame state of a layer. Certain actions must only be performed at the start of a new frame and other actions must wait until the end of the current frame.
	laContextFrameState_t	Possible values for context frame state.
	laContextFrameState	Possible values for context frame state.
	laImageWidget_DrawEventCallback	
	laContextUpdateState_t	Possible values for context update state.
	laWidgetUpdateState_t	Specifies the different update states the widget can be assigned
	laContextUpdateState	Possible values for context update state.
	laWidgetUpdateState	Specifies the different update states the widget can be assigned

Description

This section Aria User Interface Library Interface.

a) Functions

laContext_AddScreen Function

Add screen to the list of screens in the current context

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_AddScreen(laScreen* screen);
```

Returns

laResult

Description

Add screen to the list of screens in the current context

Function

laResult laContext_AddScreen(laScreen* screen)

laContext_Create Function

Creates an instance of the Aria user interface library

File

libaria_context.h

C

```
LIB_EXPORT laContext* laContext_Create(GFX_Driver driver, GFX_Display display, GFX_Processor processor,
GFX_ColorMode mode, GFXU_MemoryIntf* memoryIntf);
```

Returns

laContext* - a valid context pointer or NULL

Preconditions

Should have called laInitialize() before attempting to create a context

Parameters

Parameters	Description
GFX_Driver	the graphics controller the library will initialize the HAL with
GFX_Display	the graphics display the library will initialize the HAL with
GFX_ColorMode	the color mode the library will use and initialize the HAL with
GFXU_MemoryIntf*	the memory interface the library will use and will initialize the HAL with

Function

laContext* laContext_Create(laArray*)

laContext_Destroy Function

Destroys an Aria instance

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_Destroy(laContext* context);
```

Returns

laResult - indicates if the instance was successfully destroyed

Parameters

Parameters	Description
laContext*	a valid Aria pointer

Function

laResult laContext_Destroy(laContext*)

laContext_GetActive Function

Returns the current active context.

File

libaria_context.h

C

```
LIB_EXPORT laContext* laContext_GetActive();
```

Returns

laContext*

Function

laContext* laContext_GetActive()

laContext_GetActiveScreen Function

Returns the active screen of the current context

File

libaria_context.h

C

```
LIB_EXPORT laScreen* laContext_GetActiveScreen();
```

Returns

laScreen*

Description

Returns the active screen of the current context

Function

laScreen* laContext_GetActiveScreen()

laContext_GetActiveScreenIndex Function

Return the index of the active screen

File

libaria_context.h

C

```
LIB_EXPORT int32_t laContext_GetActiveScreenIndex();
```

Returns

int32_t

Description

Return the index of the active screen

Function

int32_t laContext_GetActiveScreenIndex()

laContext_GetColorMode Function

Returns the color mode of the current context

File

libaria_context.h

C

```
LIB_EXPORT GFX_ColorMode laContext_GetColorMode();
```

Returns

GFX_ColorMode

Function

GFX_ColorMode laContext_GetColorMode()

laContext_GetDefaultScheme Function

Returns the pointer to the default scheme of the current context

File

libaria_context.h

C

```
LIB_EXPORT laScheme* laContext_GetDefaultScheme();
```

Returns

laScheme*

Description

Returns the pointer to the default scheme of the current context

Function

laScheme* laContext_GetDefaultScheme()

laContext_GetEditWidget Function

Gets the widget that is currently recieving all widget edit events.

File

libaria_context.h

C

```
LIB_EXPORT laEditWidget* laContext_GetEditWidget();
```

Returns

laEditWidget*

Description

Edit widgets are widgets that inherit the 'edit widget' API function list. These widgets are capable of receiving edit events from other widgets that are edit event broadcasters. A broadcaster could be a 'key pad' and a receiver could be a 'text edit' box.

Function

laEditWidget* laContext_GetEditWidget()

laContext_GetFocusWidget Function

Return a pointer to the widget in focus

File

libaria_context.h

C

```
LIB_EXPORT laWidget* laContext_GetFocusWidget();
```

Returns

laWidget*

Description

The focus widget is the widget that is currently receiving all input events. This can happen when the user initiates a touch down event on the widget and is currently dragging their finger on the display. The widget will receive all touch moved events until a touch up event is received.

Function

laWidget* laContext_GetFocusWidget()

laContext_GetPremptionLevel Function

Returns the preemption level for the screen

File

libaria_context.h

C

```
LIB_EXPORT laPreemptionLevel laContext_GetPremptionLevel();
```

Returns

laPreemptionLevel

Description

Returns the preemption level for the screen

Function

laPreemptionLevel laContext_GetPreemptionLevel()

laContext_GetScreenRect Function

Returns the display rectangle structure of the physical display

File

libaria_context.h

C

```
LIB_EXPORT GFX_Rect laContext_GetScreenRect();
```

Returns

GFX_Rect

Description

Returns the display rectangle - width height and upper left corner coordinates of the physical display

Function

LIB_EXPORT GFX_Rect laContext_GetScreenRect()

laContext_GetStringLanguage Function

Returns the language index of the current context

File

libaria_context.h

C

```
LIB_EXPORT uint32_t laContext_GetStringLanguage();
```

Returns

uint32_t

Description

Returns the language index of the current context

Function

uint32_t laContext_GetStringLanguage()

laContext_GetStringTable Function

Get a pointer to the GFXU_StringTableAsset structure that maintains the strings, associated fonts, etc

File

libaria_context.h

C

```
LIB_EXPORT GFXU_StringTableAsset* laContext_GetStringTable();
```

Returns

GFXU_StringTableAsset*

Description

Get a pointer to the GFXU_StringTableAsset structure that maintains the strings, associated fonts, etc

Function

GFXU_StringTableAsset* laContext_GetStringTable()

laContext_HideActiveScreen Function

Hide the active screen

File

libaria_context.h

C

```
LIB_EXPORT GFX_DEPRECATED laResult laContext_HideActiveScreen();
```

Returns

void

Description

Hide the active screen. If the screen's persistent flag is set to true then the memory for the screen's widgets will not be deallocated. This will maintain the state of the screen.

Function

laResult laContext_HideActiveScreen()

laContext_RedrawAll Function

Forces the library to redraw the currently active screen in its entirety.

File

libaria_context.h

C

```
LIB_EXPORT void laContext_RedrawAll();
```

Returns

void

Function

void laContext_RedrawAll()

laContext_RemoveScreen Function

Remove the specified screen from the list of screens in the current context

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_RemoveScreen(laScreen* screen);
```

Returns

laResult

Description

Remove the specified screen from the list of screens in the current context

Function

laResult laContext_RemoveScreen(laScreen* screen)

laContext_SetActive Function

Make the specified context active

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetActive(laContext* context);
```

Returns

laResult - LA_SUCCESS if the context was successfully set as active

Function

laResult laContext_SetActive(laContext* context)

laContext_SetActiveScreen Function

Change the active screen to the one specified by the index argument

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetActiveScreen(uint32_t id);
```

Returns

void

Description

This operation will tear down the existing layer state of the driver if necessary and rebuild the frame buffers if the existing buffers can not be reused. This operation can be potentially slow and expensive. Widgets can be used to simulate screen transitions as applicable.

Function

laResult laContext_SetActiveScreen(uint32_t id)

laContext_SetActiveScreenChangedCallback Function

Set the callback function pointer when the screen change event occurs

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetActiveScreenChangedCallback(laContext_ActiveScreenChangedCallback_FnPtr
cb);
```

Returns

laResult

Description

Set the callback function pointer when the screen change event occurs

Function

laResult laContext_SetActiveScreenChangedCallback(laContext_ActiveScreenChangedCallback_FnPtr cb)

laContext_SetEditWidget Function

Sets the currently active edit widget.

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetEditWidget(laWidget* widget);
```

Returns

laResult

Parameters

Parameters	Description
laWidget*	a widget that inherits the edit widget API and has its 'editable' flag set to true.

Function

laResult laContext_SetEditWidget(laWidget* widget)

laContext_SetFocusWidget Function

Set into focus the widget specified as the argument

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetFocusWidget(laWidget* widget);
```

Returns

laResult

Description

Set into focus the widget specified as the argument

Function

laResult laContext_SetFocusWidget(laWidget* widget)

laContext_SetLanguageChangedCallback Function

Set the callback function pointer when the language change event occurs

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetLanguageChangedCallback(laContext_LanguageChangedCallback_FnPtr cb);
```

Returns

laResult

Description

Set the callback function pointer when the language change event occurs

Function

laResult laContext_SetLanguageChangedCallback(laContext_LanguageChangedCallback_FnPtr cb)

laContext_SetStringLanguage Function

Set the language index of the current context

File

libaria_context.h

C

```
LIB_EXPORT void laContext_SetStringLanguage(uint32_t id);
```

Returns

void

Description

Set the language index of the current context

Function

void laContext_SetStringLanguage(uint32_t id)

laContext_SetStringTable Function

Set the StringTable pointer to the specified new StringTableAsset structure

File

libaria_context.h

C

```
LIB_EXPORT void laContext_SetStringTable(GFXU_StringTableAsset* table);
```

Returns

void

Description

Set the StringTable pointer to the specified new StringTableAsset structure

Function

void laContext_SetStringTable(GFXU_StringTableAsset* table)

laContext_Update Function

Runs the update loop for a library instance.

File

libaria_context.h

C

```
LIB_EXPORT void laContext_Update(uint32_t dt);
```

Returns

void

Description

The update loop allows the library to service its event array and allows any intelligent widgets to perform active update tasks. This should be run periodically, but not often enough to starve other processes. Running too little may result in a loss of UI responsiveness.

Parameters

Parameters	Description
uint32_t dt	a delta time representing how much time has passed since the last time laContext_Update has been called. This is typically in milliseconds.

Function

void laContext_Update(uint32_t dt)

laEditWidget_Accept Function

File

libaria_editwidget.h

C

```
void laEditWidget_Accept();
```

Description

This is function laEditWidget_Accept.

laEditWidget_Append Function

File

libaria_editwidget.h

C

```
void laEditWidget_Append(laString str);
```

Description

This is function laEditWidget_Append.

laEditWidget_Backspace Function

File

libaria_editwidget.h

C

```
void laEditWidget_Backspace();
```

Description

This is function laEditWidget_Backspace.

laEditWidget_Clear Function

File

libaria_editwidget.h

C

```
void laEditWidget_Clear();
```

Description

This is function laEditWidget_Clear.

laEditWidget_EndEdit Function

File

libaria_editwidget.h

C

```
void laEditWidget_EndEdit();
```

Description

This is function laEditWidget_EndEdit.

laEditWidget_Set Function

File

libaria_editwidget.h

C

```
void laEditWidget_Set(laString str);
```

Description

This is function laEditWidget_Set.

laEditWidget_StartEdit Function

File

libaria_editwidget.h

C

```
laResult laEditWidget_StartEdit();
```

Description

This is function laEditWidget_StartEdit.

laList_Assign Function

Assignes a new pointer to an index in the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_Assign(laList* list, size_t idx, void* val);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
size_t idx	the index to modify
void* val	the new value of the node

Function

int32_t laList_Assign(laList* list, size_t idx, void* val)

laList_Clear Function

Removes all nodes from a given list

File

libaria_list.h

C

```
LIB_EXPORT void laList_Clear(laList* list);
```

Returns

void

Parameters

Parameters	Description
laList* list	the list to modify

Function

void laList_Clear(laList* list)

laList_Copy Function

Creates a duplicate of an existing list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_Copy(laList* l, laList* r);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* l	the source list
laList* r	the result list

Function

int32_t laList_Copy(laList* l, laList* r)

laList_Create Function

Initializes a new linked list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_Create(laList* list);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to initilaize

Function

int32_t laList_Create(laList* list)

laList_Destroy Function

Removes all nodes from a given list and frees the data of each node

File

libaria_list.h

C

```
LIB_EXPORT void laList_Destroy(laList* list);
```

Returns

void

Parameters

Parameters	Description
laList* list	the list to modify

Function

void laList_Destroy(laList* list)

laList_Find Function

Retrieves the index of a value from the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_Find(laList* list, void* val);
```

Returns

int32_t - the index of the value searched for

Parameters

Parameters	Description
laList* list	pointer to the list to reference
void* val	the value to search for

Function

int32_t laList_Find(laList* list, void* val)

laList_Get Function

Retrieves a value from the list

File

libaria_list.h

C

```
LIB_EXPORT void* laList_Get(laList* list, uint32_t idx);
```

Returns

void* - the retrieved value

Parameters

Parameters	Description
laList* list	pointer to the list to reference
uint32_t idx	the index of the value to retrieve

Function

void* laList_Get(laList* list, uint32_t idx)

laList_InsertAt Function

Inserts an item into a list at a given index. All existing from index are shifted right one place.

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_InsertAt(laList* list, void* val, uint32_t idx);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
void* val	the value to insert
uint32_t idx	the position to insert the value

Function

int32_t laList_InsertAt(laList* list,
void* val,
uint32_t idx)

laList_PopBack Function

Removes the last value from the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_PopBack(laList* list);
```

Parameters

Parameters	Description
laList* list	pointer to the list to modify

Function

void laList_PopBack(laList* list)

laList_PopFront Function

Removes the first value from the list

File

libaria_list.h

C

```
LIB_EXPORT void laList_PopFront(laList* list);
```

Parameters

Parameters	Description
laList* list	pointer to the list to modify

Function

void laList_PopFront(laList* list)

laList_PushBack Function

Pushes a new node onto the back of the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_PushBack(laList* list, void* val);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
void* val	the new value of the node

Function

int32_t laList_PushBack(laList* list, void* val)

laList_PushFront Function

Pushes a new node onto the front of the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_PushFront(laList* list, void*);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
void* val	the new value of the node

Function

int32_t laList_PushFront(laList* list, void* val)

laList_Remove Function

Removes an item from the list

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_Remove(laList* list, void*);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
void* val	the value to remove

Function

int32_t laList_Remove(laList* list, void*)

laList_RemoveAt Function

Removes an item from the list at an index

File

libaria_list.h

C

```
LIB_EXPORT int32_t laList_RemoveAt(laList* list, uint32_t idx);
```

Returns

int32_t - 0 if success, -1 if failure

Parameters

Parameters	Description
laList* list	pointer to the list to modify
uint32_t idx	the index of the value to remove

Function

int32_t laList_Remove(laList* list, uint32_t idx)

laString_Allocate Function

Attempts to resize the local data buffer for a string.

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_Allocate(laString* str, uint32_t size);
```

Returns

laResult - LA_SUCCESS if the function succeeded

Remarks

If size is zero then the memory will be freed and the function will return success.

Parameters

Parameters	Description
laString* str	the string to modify
uint32_t size	the desired size of the string

Function

void laString_Allocate(laString* str, uint32_t size)

laString_Append Function

Appends a string onto the end of another string

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_Append(laString* dst, const laString* src);
```

Returns

laResult - LA_SUCCESS if the operation succeeded

Parameters

Parameters	Description
laString* dst	the destination string
const laString* src	the source string

Function

void laString_Append(laString* dst, const laString* src)

laString_Capacity Function

Returns the capacity of a string

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_Capacity(const laString* str);
```

Returns

uint32_t - the capacity of a string in characters

Parameters

Parameters	Description
const laString* str	the string to reference

Function

uint32_t laString_Capacity(const laString* str)

laString_CharAt Function

Extracts the code point for the character in a string at a given index.

File

libaria_string.h

C

```
LIB_EXPORT GFXU_CHAR laString_CharAt(const laString* str, uint32_t idx);
```

Returns

GFXU_CHAR - the code point of the character

Parameters

Parameters	Description
const laString* str	the string to reference
uint32_t idx	the character index to reference

Function

GFXU_CHAR laString_CharAt(const laString* str, uint32_t idx)

laString_Clear Function

Sets a string's length to zero and its string table reference to NULL. Does not free any associated data and preserves capacity.

File

libaria_string.h

C

```
LIB_EXPORT void laString_Clear(laString* str);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to modify

Function

void laString_Clear(laString* str)

laString_Compare Function

Compares two string objects

File

libaria_string.h

C

```
LIB_EXPORT int32_t laString_Compare(const laString* lstr, const laString* rstr);
```

Returns

int32_t - the result of the string comparison, 0 if the strings are equal see strcmp() for more information

Parameters

Parameters	Description
const laString* lstr	the left argument
const laString* rstr	the right argument

Function

int32_t laString_Compare(const laString* lstr, const laString* rstr)

laString_CompareBuffer Function

Compares a string object and a GFXU_CHAR* buffer

File

libaria_string.h

C

```
LIB_EXPORT int32_t laString_CompareBuffer(const laString* str, const GFXU_CHAR* buffer);
```

Returns

int32_t - the result of the string comparison, 0 if the strings are equal see strcmp() for more information

Parameters

Parameters	Description
const laString* lstr	the string
const GFXU_CHAR* buffer	the GFXU_CHAR buffer

Function

int32_t laString_Compare(const laString* lstr, const GFXU_CHAR* buffer)

laString_Copy Function

Copies the values from one string into another

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_Copy(laString* dst, const laString* src);
```

Returns

laResult - LA_SUCCESS if the function succeeded

Remarks

Makes duplicate of a given string. Destination will have the same length and data but may not have the same overall capacity. The source may have lots of unused space and the destination may not match to avoid waste. Caller is responsible for the allocated memory but does not need to preserve the input string to maintain the destination string buffer state.

Caller must also ensure that the font contains all the glyphs for the string or rendering may not be possible.

Parameters

Parameters	Description
laString* dst	the destination string object
laString* src	the source string object

Function

laResult laString_Copy(laString* dst, const laString* src)

laString_CreateFromBuffer Function

Creates a string object from a GFXU_CHAR buffer and a font asset pointer

File

libaria_string.h

C

```
LIB_EXPORT laString laString_CreateFromBuffer(const GFXU_CHAR* chr, GFXU_FontAsset* fnt);
```

Returns

laString - created string object

Remarks

Makes an internal copy of the input buffer for the string object. Caller is responsible for the allocated memory but does not need to preserve the input buffer to maintain the string buffer state.

Caller must also ensure that the font contains all the glyphs for the string or rendering may not be possible.

Parameters

Parameters	Description
const GFXU_CHAR* chr	pointer to a GFXU_CHAR buffer, can be NULL
GFXU_FontAsset* fnt	pointer to a font asset, can be NULL

Function

laString laString_CreateFromBuffer(const GFXU_CHAR* chr, GFXU_FontAsset* fnt)

laString_CreateFromCharBuffer Function

Creates a string object from a const char* buffer and a font asset pointer. This method provides compatibility with standard c-style strings. Input string will be converted from 8-bit with to 32-bit width.

File

libaria_string.h

C

```
LIB_EXPORT laString laString_CreateFromCharBuffer(const char* chr, GFXU_FontAsset* fnt);
```

Returns

laString - created string object

Remarks

Makes an internal copy of the input buffer for the string object. Caller is responsible for the allocated memory but does not need to preserve the input buffer to maintain the string buffer state.

Caller must also ensure that the font contains all the glyphs for the string or rendering may not be possible.

Parameters

Parameters	Description
const char* chr	pointer to a const char* buffer, can be NULL
GFXU_FontAsset* fnt	pointer to a font asset, can be NULL

Function

laString laString_CreateFromCharBuffer(const char* chr, GFXU_FontAsset* fnt)

laString_CreateFromID Function

Creates a string object that simply references a string in the string table.

File

libaria_string.h

C

```
LIB_EXPORT laString laString_CreateFromID(uint32_t id);
```

Returns

laString - created string object

Remarks

Allocates no memory.

Parameters

Parameters	Description
uint32_t id	the string table id to use

Function

laString laString_CreateFromID(uint32_t id)

laString_Delete Function

Deletes all memory associated with a string object

File

libaria_string.h

C

```
LIB_EXPORT void laString_Delete(laString** str);
```

Returns

void

Remarks

Will free local string data and the memory for the string pointer itself, setting the pointer to NULL if successful

Parameters

Parameters	Description
laString** str	pointer to a pointer to a string object

Function

void laString_Delete(laString** str)

laString_Destroy Function

Destroys a string object. This frees the strings internal data buffer, if it exists, sets its string table reference to null, and clears all supporting attributes.

File

libaria_string.h

C

```
LIB_EXPORT void laString_Destroy(laString* str);
```

Parameters

Parameters	Description
laString* str	the string to modify

Function

void laString_Destroy(laString* str)

laString_Draw Function

Wrapper around GFX Utility string draw function for Aria user interface library. Internal use only.

File

libaria_string.h

C

```
LIB_EXPORT void laString_Draw(laString* str, int32_t x, int32_t y, GFXU_ExternalAssetReader** reader);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to draw
int32_t x	x position to render at
int32_t y	y position to render at
GFXU_ExternalAssetReader** reader	external reader state machine, if string font is located external

Function

void laString_Draw(laString* str,
int32_t x,
int32_t y,
GFXU_ExternalAssetReader** reader)

laString_ExtractFromTable Function

Extracts a read-only string from the string table into a modifiable string object. This relies on the active context to indicate which string table to reference as well as which language entry to extract.

File

libaria_string.h

C

```
LIB_EXPORT void laString_ExtractFromTable(laString* dst, uint32_t table_index);
```

Returns

void

Remarks

Caller is responsible for the allocated memory but does not need to preserve the input buffer to maintain the string buffer state.

Parameters

Parameters	Description
laString* dst	the destination string object
uint32_t table_index	the table index to extract

Function

void laString_ExtractFromTable(laString* dst, uint32_t table_index)

laString_GetCharIndexAtPoint Function

Given an offset in pixels returns the corresponding character index.

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_GetCharIndexAtPoint(laString* str, int32_t x);
```

Returns

uint32_t - character index

Parameters

Parameters	Description
laString* str	the string to reference
int32_t x	x offset in pixels

Function

uint32_t laString_GetCharIndexAtPoint(laString* str, int32_t x)

laString_GetCharOffset Function

Returns the offset of a given character index in pixels.

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_GetCharOffset(laString* str, uint32_t idx);
```

Returns

uint32_t - the offset in pixels

Parameters

Parameters	Description
laString* str	the string to reference
uint32_t idx	the character index offset to calculate

Function

uint32_t laString_GetCharOffset(laString* str, uint32_t idx)

laString_GetCharWidth Function

Given a character index, gets the width of that character. Only accurate if the string has a font associated with it and that font contains all the characters in the string in question.

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_GetCharWidth(laString* str, uint32_t idx);
```

Returns

uint32_t - character width in pixels

Parameters

Parameters	Description
laString* str	the string to reference
uint32_t x	character index to reference

Function

uint32_t laString_GetCharWidth(laString* str, uint32_t idx)

laString_GetHeight Function

Returns the height of a string by referencing its associated font asset data.

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_GetHeight(laString* str);
```

Returns

uint32_t - the height of the string

Parameters

Parameters	Description
laString* str	the string to reference

Function

uint32_t laString_GetHeight(laString* str)

laString_GetRect Function

Calculates the rectangle for a given string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup.

File

libaria_string.h

C

```
LIB_EXPORT void laString_GetRect(laString* str, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to reference
GFX_Rect* rect	the calculated string rectangle result

Function

void laString_GetRect(laString* str, GFX_Rect* rect)

laString_Initialize Function

Initializes a string struct to default

File

libaria_string.h

C

```
LIB_EXPORT void laString_Initialize(laString* str);
```

Returns

void

Remarks

Allocates no memory

Parameters

Parameters	Description
laString* str	pointer to a string object

Function

void laString_Initialize(laString* str)

laString_Insert Function

Inserts a string into another string at a given index

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_Insert(laString* dst, const laString* src, uint32_t idx);
```

Returns

laResult - LA_SUCCESS if the operation succeeded

Parameters

Parameters	Description
laString* dst	the destination string
const laString* src	the source string
uint32_t idx	the insertion index

Function

void laString_Insert(laString* dst,const laString* src, uint32_t idx)

laString_Length Function

Calculates the length of a string in characters

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_Length(const laString* str);
```

Returns

uint32_t - the number of characters in the string

Parameters

Parameters	Description
const laString* str	the string to reference

Function

uint32_t laString_Length(const laString* str)

laString_New Function

Allocates a memory for a new string

File

libaria_string.h

C

```
LIB_EXPORT laString* laString_New(laString* src);
```

Returns

laString* - pointer to the newly allocated string

Remarks

Caller is responsible for freeing the memory allocated by this function

Parameters

Parameters	Description
laString* src	a string to copy, can be NULL

Function

laString* laString_New(laString* src)

laString_ReduceLength Function

Reduces the length of a string. This simply slides the null terminator to the left and does not affect the string's capacity value.

File

libaria_string.h

C

```
LIB_EXPORT void laString_ReduceLength(laString* str, uint32_t length);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to modify
uint32_t length	the new desired length in characters

Function

void laString_ReduceLength(laString* str, uint32_t length)

laString_Set Function

Attempts to set the local data buffer of a string to an input buffer

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_Set(laString* str, const GFXU_CHAR* buffer);
```

Returns

laResult - LA_SUCCESS if the function succeeded

Remarks

Makes an internal copy of the input buffer for the string object. Caller is responsible for the allocated memory but does not need to preserve the input buffer to maintain the string buffer state.

Caller must also ensure that the font contains all the glyphs for the string or rendering may not be possible.

Parameters

Parameters	Description
laString* str	the string to modify
const GFXU_CHAR* buffer	the input buffer

Function

laResult laString_Set(laString* str, const GFXU_CHAR* buffer)

laString_SetCapacity Function

Attempts to adjust the capacity of a string

File

libaria_string.h

C

```
LIB_EXPORT laResult laString_SetCapacity(laString* str, uint32_t cap);
```

Returns

laResult - LA_SUCCESS if the operation succeeded

Parameters

Parameters	Description
laString* str	the string to modify
uint32_t cap	the new desired capacity

Function

void laString_SetCapacity(laString* str, uint32_t cap)

laString_ToCharBuffer Function

Extracts the data buffer from a string and copies it into the provided buffer argument.

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_ToCharBuffer(const laString* str, GFXU_CHAR* buffer, uint32_t size);
```

Returns

uint32_t - the number of characters copied

Parameters

Parameters	Description
laString* str	the string to reference
GFXU_CHAR* buffer	the destination buffer
uint32_t size	the max size of the destination buffer

Function

uint32_t laString_ToCharBuffer(const laString* str,
GFXU_CHAR* buffer,
uint32_t size)

laButtonWidget_GetHAlignment Function

Gets the horizontal alignment setting for a button

File

libaria_widget_button.h

C

```
LIB_EXPORT laHAlignment laButtonWidget_GetHAlignment(laButtonWidget* btn);
```

Returns

laHAlignment - the horizontal alignment value

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

laHAlignment laButtonWidget_GetHAlignment(laButtonWidget* btn)

laButtonWidget_GetImageMargin Function

Gets the distance between the icon and the text

File

libaria_widget_button.h

C

```
LIB_EXPORT uint32_t laButtonWidget_GetImageMargin(laButtonWidget* btn);
```

Returns

uint32_t - the distance value

Parameters

Parameters	Description
laButtonWidget* btn	the widget

Function

uint32_t laButtonWidget_GetImageMargin(laButtonWidget* btn)

laButtonWidget_GetImagePosition Function

Gets the position of the button icon

File

libaria_widget_button.h

C

```
LIB_EXPORT laRelativePosition laButtonWidget_GetImagePosition(laButtonWidget* btn);
```

Returns

laRelativePosition

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

laRelativePosition laButtonWidget_GetImagePosition(laButtonWidget* btn)

laButtonWidget_GetPressed Function

Gets the pressed state of a button

File

libaria_widget_button.h

C

```
LIB_EXPORT laBool laButtonWidget_GetPressed(laButtonWidget* btn);
```

Returns

laBool - the button pressed state

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

laBool laButtonWidget_GetPressed(laButtonWidget* btn)

laButtonWidget_GetPressedEventCallback Function

Gets the callback associated with the button pressed event

File

libaria_widget_button.h

C

```
LIB_EXPORT laButtonWidget_PressedEvent laButtonWidget_GetPressedEventCallback(laButtonWidget* btn);
```

Returns

laButtonWidget_PressedEvent

Parameters

Parameters	Description
laButtonWidget* btn	the widget

Function

laButtonWidget_PressedEvent laButtonWidget_GetPressedEventCallback(laButtonWidget* btn)

laButtonWidget_GetPressedImage Function

Gets the pressed image asset pointer for a button

File

libaria_widget_button.h

C

```
LIB_EXPORT GFXU_ImageAsset* laButtonWidget_GetPressedImage(laButtonWidget* btn);
```

Returns

GFXU_ImageAsset* - the pressed asset pointer

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

GFXU_ImageAsset* laButtonWidget_GetPressedImage(laButtonWidget* btn)

laButtonWidget_GetPressedOffset Function

Gets the offset of the button internals when pressed

File

libaria_widget_button.h

C

```
LIB_EXPORT int32_t laButtonWidget_GetPressedOffset(laButtonWidget* btn);
```

Returns

int32_t - the distance value

Parameters

Parameters	Description
laButtonWidget* btn	the widget

Function

int32_t laButtonWidget_GetPressedOffset(laButtonWidget* btn)

laButtonWidget_GetReleasedEventCallback Function

Gets the callback for the button released event

File

libaria_widget_button.h

C

```
LIB_EXPORT laButtonWidget_ReleasedEvent laButtonWidget_GetReleasedEventCallback(laButtonWidget* btn);
```

Returns

laButtonWidget_ReleasedEvent

Parameters

Parameters	Description
laButtonWidget* btn	the widget

Function

laButtonWidget_ReleasedEvent laButtonWidget_GetReleasedEventCallback(laButtonWidget* btn)

laButtonWidget_GetReleasedImage Function

Gets the currently used released icon

File

libaria_widget_button.h

C

```
LIB_EXPORT GFXU_ImageAsset* laButtonWidget_GetReleasedImage(laButtonWidget* btn);
```

Returns

GFXU_ImageAsset* - the released asset pointer

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

GFXU_ImageAsset* laButtonWidget_GetReleasedImage(laButtonWidget* btn)

laButtonWidget_GetText Function

Gets the text for a button. If the button's string has local data then a duplicate of the string will be allocated. The caller is responsible for managing the memory for the duplicated string. If the button string is a string table reference then only the reference ID is copied.

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_GetText(laButtonWidget* btn, laString* str);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference
laString* str	pointer to a string to copy the button string into

Function

laResult laButtonWidget_GetText(laButtonWidget* btn, laString* str)

laButtonWidget_GetToggleable Function

Gets the value of this button's toggle flag

File

libaria_widget_button.h

C

```
LIB_EXPORT laBool laButtonWidget_GetToggleable(laButtonWidget* btn);
```

Returns

laBool - the value of the toggle flag

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

laBool laButtonWidget_GetToggleable(laButtonWidget* btn)

laButtonWidget_GetVAlignment Function

Gets the vertical alignment setting for a button

File

libaria_widget_button.h

C

```
LIB_EXPORT laVAlignment laButtonWidget_GetVAlignment(laButtonWidget* btn);
```

Returns

laVAlignment - the vertical alignment setting for the button

Parameters

Parameters	Description
laButtonWidget* btn	the button to reference

Function

laVAlignment laButtonWidget_GetVAlignment(laButtonWidget* btn)

laButtonWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_button.h

C

```
LIB_EXPORT laButtonWidget* laButtonWidget_New();
```

Returns

laButtonWidget* - pointer to a new button widget instance

Description

Creates a new button widget instance. Invokes the button constructor

Remarks

Caller is responsible for managing the memory allocated by this function until the widget is added to a valid widget tree.

Function

laButtonWidget* laButtonWidget_New()

laButtonWidget_SetHAlignment Function

Sets the horizontal alignment value for a button

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetHAlignment(laButtonWidget* btn, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the button to modify
laHAlignment align	the desired alignment value

Function

laResult laButtonWidget_SetHAlignment(laButtonWidget* btn,
laHAlignment align)

laButtonWidget_SetImageMargin Function

Sets the distance between the icon and text

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetImageMargin(laButtonWidget* btn, uint32_t mg);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
uint32_t	the distance value

Function

laResult laButtonWidget_SetImageMargin(laButtonWidget* btn,
uint32_t mg)

laButtonWidget_SetImagePosition Function

Sets the position of the button icon

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetImagePosition(laButtonWidget* btn, laRelativePosition pos);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
laRelativePosition pos	the desired image position

Function

laResult laButtonWidget_SetImagePosition(laButtonWidget* btn,
laRelativePosition pos)

laButtonWidget_SetPressed Function

Sets the pressed state for a button.

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetPressed(laButtonWidget* btn, laBool pressed);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the button to modify
laBool pressed	the pressed state

Function

laResult laButtonWidget_SetPressed(laButtonWidget* btn, laBool pressed)

laButtonWidget_SetPressedEventCallback Function

Sets the pressed event callback for the button

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetPressedEventCallback(laButtonWidget* btn, laButtonWidget_PressedEvent
cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
laButtonWidget_PressedEvent cb	a valid callback pointer or NULL

Function

laResult laButtonWidget_SetPressedEventCallback(laButtonWidget* btn,
laButtonWidget_PressedEvent cb)

laButtonWidget_SetPressedImage Function

Sets the image to be used as a pressed icon

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetPressedImage(laButtonWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
GFXU_ImageAsset* img	pointer to an image asset

Function

laResult laButtonWidget_SetPressedImage(laButtonWidget* btn,
GFXU_ImageAsset* img)

laButtonWidget_SetPressedOffset Function

Sets the offset of the button internals when pressed

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetPressedOffset(laButtonWidget* btn, int32_t offs);
```

Returns

laResult - the operation result

Description

This value will be applied to all of the contents of the button when it is pressed. This helps to visualize the button being pressed.

Parameters

Parameters	Description
laButtonWidget* btn	the widget
int32_t	the distance value

Function

laResult laButtonWidget_SetPressedOffset(laButtonWidget* btn, int32_t offs)

laButtonWidget_SetReleasedEventCallback Function

Sets the callback for the button released event

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetReleasedEventCallback(laButtonWidget* btn,
laButtonWidget_ReleasedEvent cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
laButtonWidget_ReleasedEvent cb	a valid callback pointer or NULL

Function

laResult laButtonWidget_SetReleasedEventCallback(laButtonWidget* btn,
laButtonWidget_ReleasedEvent cb)

laButtonWidget_SetReleasedImage Function

Sets the image to be used as the released icon

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetReleasedImage(laButtonWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the widget
GFXU_ImageAsset* img	the image asset to be used

Function

laResult laButtonWidget_SetReleasedImage(laButtonWidget* btn,
GFXU_ImageAsset* img)

laButtonWidget_SetText Function

Sets the text for a button. If the input string has local data then the data will be copied into the button's local string, causing a memory allocation. If the input string is a string table reference then only the reference will be copied. The input string can be safely modified and the button string will not be affected.

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetText(laButtonWidget* btn, laString str);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the button to modify
laString str	the string to set to the button

Function

laResult laButtonWidget_SetText(laButtonWidget* btn, laString str)

laButtonWidget_SetToggleable Function

Enables the toggle mode for a button. When pressed, toggle buttons will stay down until pressed again.

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetToggleable(laButtonWidget* btn, laBool toggleable);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the button to modify
laBool toggleable	the desired togglestate

Function

laResult laButtonWidget_SetToggleable(laButtonWidget* btn,
laBool toggleable)

laButtonWidget_SetVAlignment Function

Sets the vertical alignment for a button

File

libaria_widget_button.h

C

```
LIB_EXPORT laResult laButtonWidget_SetVAlignment(laButtonWidget* btn, laVAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laButtonWidget* btn	the btn to modify
laVAlignment align	the desired vertical alignment setting

Function

laResult laButtonWidget_SetVAlignment(laButtonWidget* btn,
laVAlignment align)

laCheckBoxWidget_GetChecked Function

Gets the checked state of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laBool laCheckBoxWidget_GetChecked(laCheckBoxWidget* cbox);
```

Returns

laBool - the checked flag value

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laBool laCheckBoxWidget_GetChecked(laCheckBoxWidget* cbox)

laCheckBoxWidget_GetCheckedEventCallback Function

Gets the checked event callback

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laCheckBoxWidget_CheckedEvent laCheckBoxWidget_GetCheckedEventCallback(laCheckBoxWidget* cbox);
```

Returns

laCheckBoxWidget_CheckedEvent - a valid callback pointer or NULL

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laCheckBoxWidget_CheckedEvent laCheckBoxWidget_GetCheckedEventCallback(laCheckBoxWidget* cbox)

laCheckBoxWidget_GetCheckedImage Function

Gets the checked image of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT GFXU_ImageAsset* laCheckBoxWidget_GetCheckedImage(laCheckBoxWidget* btn);
```

Returns

GFXU_ImageAsset* - the current checked image asset pointer

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

GFXU_ImageAsset* laCheckBoxWidget_GetCheckedImage(laCheckBoxWidget* btn)

laCheckBoxWidget_GetHAlignment Function

Gets the horizontal alignment of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laHAlignment laCheckBoxWidget_GetHAlignment(laCheckBoxWidget* cbox);
```

Returns

laHAlignment - the current halign value

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laHAlignment laCheckBoxWidget_GetHAlignment(laCheckBoxWidget* cbox)

laCheckBoxWidget_GetImageMargin Function

Gets the distance between the image and the text

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT uint32_t laCheckBoxWidget_GetImageMargin(laCheckBoxWidget* btn);
```

Returns

uint32_t - the current image margin value

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

uint32_t laCheckBoxWidget_GetImageMargin(laCheckBoxWidget* btn)

laCheckBoxWidget_GetImagePosition Function

Gets the image position of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laRelativePosition laCheckBoxWidget_GetImagePosition(laCheckBoxWidget* btn);
```

Returns

laRelativePosition - the current image position value

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laRelativePosition laCheckBoxWidget_GetImagePosition(laCheckBoxWidget* btn)

laCheckBoxWidget_GetText Function

Gets a copy of the checkbox text. If the text has local data the data will be duplicated. The caller is responsible for managing the memory as appropriate.

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_GetText(laCheckBoxWidget* cbox, laString* str);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
str	pointer to an laString object

Function

laResult laCheckBoxWidget_GetText(laCheckBoxWidget* cbox, laString* str)

laCheckBoxWidget_GetUncheckedEventCallback Function

Gets the unchecked event callback

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laCheckBoxWidget_UncheckedEvent laCheckBoxWidget_GetUncheckedEventCallback(laCheckBoxWidget*
cbox);
```

Returns

laCheckBoxWidget_UncheckedEvent - a valid callback pointer or NULL

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laCheckBoxWidget_UncheckedEvent laCheckBoxWidget_GetUncheckedEventCallback(laCheckBoxWidget* cbox)

laCheckBoxWidget_GetUncheckedImage Function

Gets the unchecked image of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT GFXU_ImageAsset* laCheckBoxWidget_GetUncheckedImage(laCheckBoxWidget* btn);
```

Returns

GFXU_ImageAsset* - the current unchecked image asset pointer

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

GFXU_ImageAsset* laCheckBoxWidget_GetUncheckedImage(laCheckBoxWidget* btn)

laCheckBoxWidget_GetVAlignment Function

Gets the vertical alignment of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laVAlignment laCheckBoxWidget_GetVAlignment(laCheckBoxWidget* cbox);
```

Returns

laVAlignment

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laVAlignment laCheckBoxWidget_GetVAlignment(laCheckBoxWidget* cbox)

laCheckBoxWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is

added to a widget tree.

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laCheckBoxWidget* laCheckBoxWidget_New();
```

Returns

laCheckBoxWidget*

Function

laCheckBoxWidget* laCheckBoxWidget_New()

laCheckBoxWidget_SetChecked Function

Sets the checked state of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetChecked(laCheckBoxWidget* cbox, laBool checked);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
laBool checked	the desired checked value

Function

laResult laCheckBoxWidget_SetChecked(laCheckBoxWidget* cbox,
laBool checked)

laCheckBoxWidget_SetCheckedEventCallback Function

Sets the checked event callback

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetCheckedEventCallback(laCheckBoxWidget* cbox,
laCheckBoxWidget_CheckedEvent cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
laCheckBoxWidget_CheckedEvent cb	a valid callback pointer or NULL

Function

laResult laCheckBoxWidget_SetCheckedEventCallback(laCheckBoxWidget* cbox,
laCheckBoxWidget_CheckedEvent cb)

laCheckBoxWidget_SetCheckedImage Function

Sets the checked image of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetCheckedImage(laCheckBoxWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
GFXU_ImageAsset* img	the desired checked image asset pointer

Function

laResult laCheckBoxWidget_SetCheckedImage(laCheckBoxWidget* btn,
GFXU_ImageAsset* img)

laCheckBoxWidget_SetHAlignment Function

Sets the horizontal alignment of the check box.

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetHAlignment(laCheckBoxWidget* cbox, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
laHAlignment align	the desired halign value

Function

laResult laCheckBoxWidget_SetHAlignment(laCheckBoxWidget* cbox,
laHAlignment align)

laCheckBoxWidget_SetImagePosition Function

Sets the image position of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetImagePosition(laCheckBoxWidget* btn, laRelativePosition pos);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

laRelativePosition pos	the desired image position value

Function

laResult laCheckBoxWidget_SetImagePosition(laCheckBoxWidget* btn,
laRelativePosition pos)

laCheckBoxWidget_SetText Function

Sets the checkbox text to the input string. If the string has local data the data will be duplicated and copied to the checkboxes internal string.

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetText(laCheckBoxWidget* cbox, laString str);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget

Function

laResult laCheckBoxWidget_SetText(laCheckBoxWidget* cbox, laString str)

laCheckBoxWidget_SetUncheckedEventCallback Function

Sets the unchecked event callback

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetUncheckedEventCallback(laCheckBoxWidget* cbox,
laCheckBoxWidget_UncheckedEvent cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
laCheckBoxWidget_UncheckedEvent cb	a valid callback pointer or NULL

Function

laResult laCheckBoxWidget_SetUncheckedEventCallback(laCheckBoxWidget* cbox,
laCheckBoxWidget_UncheckedEvent cb)

laCheckBoxWidget_SetUncheckedImage Function

Sets the unchecked image of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetUncheckedImage(laCheckBoxWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
GFXU_ImageAsset* img	the desired unchecked image asset pointer

Function

laResult laCheckBoxWidget_SetUncheckedImage(laCheckBoxWidget* btn,
GFXU_ImageAsset* img)

laCheckBoxWidget_SetVAlignment Function

Sets the vertical alignment of the check box

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetVAlignment(laCheckBoxWidget* cbox, laVAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
laVAlignment align	the valign value

Function

laResult laCheckBoxWidget_SetVAlignment(laCheckBoxWidget* cbox,
laVAlignment align)

laCircleWidget_GetOrigin Function

Gets the origin coordiates of a circle widget

File

libaria_widget_circle.h

C

```
LIB_EXPORT laResult laCircleWidget_GetOrigin(laCircleWidget* cir, int32_t* x, int32_t* y);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCircleWidget* cir	the widget
int32_t* x	pointer to an integer pointer to store x
int32_t* y	pointer to an integer pointer to store y

Function

laResult laCircleWidget_GetOrigin(laCircleWidget* cir, int32_t* x, int32_t* y)

laCircleWidget_GetRadius Function

Gets the radius of a circle widget

File

libaria_widget_circle.h

C

```
LIB_EXPORT uint32_t laCircleWidget_GetRadius(laCircleWidget* cir);
```

Returns

uint32_t

Parameters

Parameters	Description
laCircleWidget* cir	the widget

Function

uint32_t laCircleWidget_GetRadius(laCircleWidget* cir)

laCircleWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_circle.h

C

```
LIB_EXPORT laCircleWidget* laCircleWidget_New();
```

Returns

laCircleWidget*

Function

laCircleWidget* laCircleWidget_New()

laCircleWidget_SetOrigin Function

Sets the origin coordiates of a circle widget

File

libaria_widget_circle.h

C

```
LIB_EXPORT laResult laCircleWidget_SetOrigin(laCircleWidget* cir, int32_t x, int32_t y);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCircleWidget* cir	the widget
int32_t x	the desired x origin coordinate
int32_t y	the desired y origin coordinate

Function

laResult laCircleWidget_SetOrigin(laCircleWidget* cir, int32_t x, int32_t y)

laCircleWidget_SetRadius Function

Sets the radius of a circle widget

File

libaria_widget_circle.h

C

```
LIB_EXPORT laResult laCircleWidget_SetRadius(laCircleWidget* cir, uint32_t rad);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCircleWidget* cir	the widget
uint32_t red	the desired radius value

Function

laResult laCircleWidget_SetRadius(laCircleWidget* cir, uint32_t rad)

laDraw_1x2BevelBorder Function

Internal utility function to draw a 1x2 bevel border

File

libaria_draw.h

C

```
LIB_EXPORT void laDraw_1x2BevelBorder(GFX_Rect* rect, GFX_Color topColor, GFX_Color bottomInnerColor,
GFX_Color bottomOuterColor);
```

Parameters

Parameters	Description
GFX_Rect* rect	the rect to draw (screen space)
GFX_Color topColor	the color of the top left lines
GFX_Color bottomInnerColor	the color of the bottom inner line
GFX_Color bottomOuterColor	the color of the bottom outer line

Function

void laDraw_1x2BevelBorder(GFX_Rect* rect,
GFX_Color topColor,
GFX_Color bottomInnerColor,
GFX_Color bottomOuterColor)

laDraw_2x1BevelBorder Function

Internal utility function to draw a 2x1 bevel border

File

libaria_draw.h

C

```
LIB_EXPORT void laDraw_2x1BevelBorder(GFX_Rect* rect, GFX_Color topOuterColor, GFX_Color topInnerColor,
GFX_Color bottomOuterColor);
```

Parameters

Parameters	Description
GFX_Rect* rect	the rect to draw (screen space)
GFX_Color topOuterColor	the color of the top outer line
GFX_Color topInnerColor	the color of the top inner line
GFX_Color bottomOuterColor	the color of the bottom lines

Function

void laDraw_2x1BevelBorder(GFX_Rect* rect,
GFX_Color topOuterColor,

GFX_Color topInnerColor,
GFX_Color bottomOuterColor)

laDraw_2x2BevelBorder Function

Internal utility function to draw a 2x2 bevel border

File

libaria_draw.h

C

```
LIB_EXPORT void laDraw_2x2BevelBorder(GFX_Rect* rect, GFX_Color topOuterColor, GFX_Color topInnerColor,
GFX_Color bottomInnerColor, GFX_Color bottomOuterColor);
```

Parameters

Parameters	Description
GFX_Rect* rect	the rect to draw (screen space)
GFX_Color topOuterColor	the color of the top outer line
GFX_Color topInnerColor	the color of the top inner line
GFX_Color bottomInnerColor	the color of the bottom inner line
GFX_Color bottomOuterColor	the color of the bottom outer line

Function

void laDraw_2x2BevelBorder(GFX_Rect* rect,
GFX_Color topOuterColor,
GFX_Color topInnerColor,
GFX_Color bottomInnerColor,
GFX_Color bottomOuterColor)

laDraw_LineBorder Function

Internal utility function to draw a basic line border

File

libaria_draw.h

C

```
LIB_EXPORT void laDraw_LineBorder(GFX_Rect* rect, GFX_Color color);
```

Parameters

Parameters	Description
GFX_Rect* rect	the rect to draw (screen space)
GFX_Color color	the color to draw

Function

void laDraw_LineBorder(GFX_Rect* rect, GFX_Color color)

laDrawSurfaceWidget_GetDrawCallback Function

Returns the pointer to the currently set draw callback.

File

libaria_widget_drawsurface.h

C

```
LIB_EXPORT laDrawSurfaceWidget_DrawCallback laDrawSurfaceWidget_GetDrawCallback(laDrawSurfaceWidget* sfc);
```

Returns

laDrawSurfaceWidget_DrawCallback - a valid callback pointer or NULL

Parameters

Parameters	Description
laDrawSurfaceWidget* sfc	the widget

Function

laDrawSurfaceWidget_DrawCallback laDrawSurfaceWidget_GetDrawCallback(laDrawSurfaceWidget* sfc)

laDrawSurfaceWidget_New Function

Allocates memory for a new DrawSurface widget.

File

libaria_widget_drawsurface.h

C

```
LIB_EXPORT laDrawSurfaceWidget* laDrawSurfaceWidget_New();
```

Returns

laDrawSurfaceWidget*

Description

Allocates memory for a new DrawSurface widget. The application is responsible for the managment of this memory until the widget is added to a widget tree.

Function

laDrawSurfaceWidget* laDrawSurfaceWidget_New()

laDrawSurfaceWidget_SetDrawCallback Function

Sets the draw callback pointer for the draw surface widget.

File

libaria_widget_drawsurface.h

C

```
LIB_EXPORT laResult laDrawSurfaceWidget_SetDrawCallback(laDrawSurfaceWidget* sfc,
laDrawSurfaceWidget_DrawCallback cb);
```

Returns

laResult - the result of the operation

Description

Sets the draw callback pointer for the draw surface widget. This callback will be called during Aria's paint loop and allows the application to perform HAL draw calls. The application should not adjust HAL layer, buffer, or context options in any way during this phase.

The callback should return GFX_TRUE if it has completed drawing. Returning GFX_FALSE will indicate to the renderer that the DrawSurface requires more time to draw and will call it again during the next paint loop.

Parameters

Parameters	Description
laDrawSurfaceWidget* sfc	the widget
laDrawSurfaceWidget_DrawCallback	a valid callback pointer or NULL

Function

laResult laDrawSurfaceWidget_SetDrawCallback(laDrawSurfaceWidget* sfc,
laDrawSurfaceWidget_DrawCallback cb)

laEvent_AddEvent Function

Add the mentioned event callback to the list of events maintained by the current context

File

libaria_event.h

C

```
laResult laEvent_AddEvent(laEvent* evt);
```

Returns

laResult

Description

Add the mentioned event callback to the list of events maintained by the current context

Function

laResult laEvent_AddEvent(laEvent* evt)

laEvent_ClearList Function

Clear the event list maintained by the current context.

File

libaria_event.h

C

```
laResult laEvent_ClearList();
```

Returns

laResult

Description

Clear the event list maintained by the current context.

Function

laResult laEvent_ClearList()

laEvent_GetCount Function

Returns the number of events listed in the current context

File

libaria_event.h

C

```
LIB_EXPORT uint32_t laEvent_GetCount();
```

Returns

uint32_t

Description

Returns the number of events listed in the current context

Function

uint32_t laEvent_GetCount()

laEvent_ProcessEvents Function

Processes the screen change as well as touch events

File

libaria_event.h

C

```
laResult laEvent_ProcessEvents();
```

Returns

laResult

Description

When a screen change event occurs, the specific screen change event handler has to be called as well as some generic maintenance for the screen change like destroying or hiding screen resources needs to be done. This function handles these tasks. It also handles similarly the touch events for individual widgets in the same manner.

Function

laResult laEvent_ProcessEvents()

laEvent_SetFilter Function

Set callback pointer for current context filter event

File

libaria_event.h

C

```
LIB_EXPORT laResult laEvent_SetFilter(laEvent_FilterEvent cb);
```

Returns

laResult

Description

Set callback pointer for current context filter event

Function

laResult laEvent_SetFilter(laEvent_FilterEvent cb)

laGradientWidget_GetDirection Function

Gets the gradient direction value for this widget.

File

libaria_widget_gradient.h

C

```
LIB_EXPORT laGradientWidgetDirection laGradientWidget_GetDirection(laGradientWidget* grad);
```

Returns

laGradientWidgetDirection - the current gradient direction

Parameters

Parameters	Description
laGradientWidget* grad	the widget

Function

laGradientWidgetDirection laGradientWidget_GetDirection(laGradientWidget* grad)

laGradientWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_gradient.h

C

```
LIB_EXPORT laGradientWidget* laGradientWidget_New();
```

Returns

laGradientWidget*

Function

laGradientWidget* laGradientWidget_New()

laGradientWidget_SetDirection Function

Sets the gradient direction value for this widget.

File

libaria_widget_gradient.h

C

```
LIB_EXPORT laResult laGradientWidget_SetDirection(laGradientWidget* grad, laGradientWidgetDirection dir);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laGradientWidget* grad	the widget
laGradientWidgetDirection dir	the desired gradient direction

Function

laResult laGradientWidget_SetDirection(laGradientWidget* grad, laGradientWidgetDirection dir)

laGroupBoxWidget_GetAlignment Function

Gets the horizontal alignmnet for the group box title text

File

libaria_widget_groupbox.h

C

```
LIB_EXPORT laHAlignment laGroupBoxWidget_GetAlignment(laGroupBoxWidget* box);
```

Returns

laHAlignment - the current halign value

Parameters

Parameters	Description
laGroupBoxWidget* box	the widget

Function

laHAlignment laGroupBoxWidget_GetAlignment(laGroupBoxWidget* box)

laGroupBoxWidget_GetText Function

Gets the text value for the group box.

File

libaria_widget_groupbox.h

C

```
LIB_EXPORT laResult laGroupBoxWidget_GetText(laGroupBoxWidget* box, laString* str);
```

Returns

laResult

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laGroupBoxWidget* box	the widget
laString* str	a pointer to an laString object

Function

laResult laGroupBoxWidget_GetText(laGroupBoxWidget* lbl, laString* str)

laGroupBoxWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_groupbox.h

C

```
LIB_EXPORT laGroupBoxWidget* laGroupBoxWidget_New();
```

Returns

laGroupBoxWidget*

Description

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

Function

laGroupBoxWidget* laGroupBoxWidget_New()

laGroupBoxWidget_SetAlignment Function

Sets the alignment for the group box title text

File

libaria_widget_groupbox.h

C

```
LIB_EXPORT laResult laGroupBoxWidget_SetAlignment(laGroupBoxWidget* box, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laGroupBoxWidget* box	the widget
laHAlignment	the desired halign value

Function

laResult laGroupBoxWidget_SetAlignment(laGroupBoxWidget* box,
laHAlignment align)

laGroupBoxWidget_SetText Function

Sets the text value for the group box.

File

libaria_widget_groupbox.h

C

```
LIB_EXPORT laResult laGroupBoxWidget_SetText(laGroupBoxWidget* box, laString str);
```

Returns

void

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laGroupBoxWidget* box	the widget
laString str	an laString object

Function

void laGroupBoxWidget_SetText(laGroupBoxWidget* box, laString str)

laImageSequenceWidget_GetImage Function

Gets the image asset pointer for an entry.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT GFXU_ImageAsset* laImageSequenceWidget_GetImage(laImageSequenceWidget* img, uint32_t idx);
```

Returns

GFXU_ImageAsset* - the image asset pointer

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index

Function

GFXU_ImageAsset* laImageSequenceWidget_GetImage(laImageSequenceWidget* img,
uint32_t idx)

laImageSequenceWidget_GetImageChangedEventCallback Function

Gets the image changed event callback pointer.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laImageSequenceImageChangedEvent_FnPtr
laImageSequenceWidget_GetImageChangedEventCallback(laImageSequenceWidget* img);
```

Returns

laImageSequenceImageChangedEvent_FnPtr - a valid callback pointer or NULL

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laImageSequenceImageChangedEvent_FnPtr laImageSequenceWidget_GetImageChangedEventCallback(laImageSequenceWidget* img)

laImageSequenceWidget_GetImageCount Function

Gets the number of image entries for this widget.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT uint32_t laImageSequenceWidget_GetImageCount(laImageSequenceWidget* img);
```

Returns

uint32_t - the number of entries for this sequence widget

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

uint32_t laImageSequenceWidget_GetImageCount(laImageSequenceWidget* img)

laImageSequenceWidget_GetImageDelay Function

Gets the image delay for an entry.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT uint32_t laImageSequenceWidget_GetImageDelay(laImageSequenceWidget* img, uint32_t idx);
```

Returns

uint32_t - the delay value

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index

Function

uint32_t laImageSequenceWidget_GetImageDelay(laImageSequenceWidget* img,
uint32_t idx)

laImageSequenceWidget_GetImageHAlignment Function

Gets the horizontal alignment for an image entry

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laHAlignment laImageSequenceWidget_GetImageHAlignment(laImageSequenceWidget* img, uint32_t idx);
```

Returns

laHAlignment - the halign value

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index

Function

laHAlignment laImageSequenceWidget_GetImageHAlignment(laImageSequenceWidget* img,
uint32_t idx)

laImageSequenceWidget_GetImageVAlignment Function

Sets the vertical alignment for an image entry

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laVAlignment laImageSequenceWidget_GetImageVAlignment(laImageSequenceWidget* img, uint32_t idx);
```

Returns

laVAlignment - the valign value

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index

Function

laVAlignment laImageSequenceWidget_GetImageVAlignment(laImageSequenceWidget* img,
uint32_t idx)

laImageSequenceWidget_GetRepeat Function

Indicates if the widget will repeat through the image entries.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laBool laImageSequenceWidget_GetRepeat(laImageSequenceWidget* img);
```

Returns

laBool - indicates if the widget is automatically repeating

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laBool laImageSequenceWidget_GetRepeat(laImageSequenceWidget* img)

laImageSequenceWidget_IsPlaying Function

Indicates if the widget is currently cycling through the image entries.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laBool laImageSequenceWidget_IsPlaying(laImageSequenceWidget* img);
```

Returns

laBool - indicates if the widget is automatically cycling

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laBool laImageSequenceWidget_IsPlaying(laImageSequenceWidget* img)

laImageSequenceWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laImageSequenceWidget* laImageSequenceWidget_New();
```

Returns

laImageSequenceWidget*

Description

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

Function

laImageSequenceWidget* laImageSequenceWidget_New()

laImageSequenceWidget_Play Function

Starts the widget automatically cycling through the image entries.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_Play(laImageSequenceWidget* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laResult laImageSequenceWidget_Play(laImageSequenceWidget* img)

laImageSequenceWidget_Rewind Function

Resets the current image sequence display index to zero.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_Rewind(laImageSequenceWidget* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laResult laImageSequenceWidget_Rewind(laImageSequenceWidget* img)

laImageSequenceWidget_SetImage Function

Sets the image asset pointer for an entry.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImage(laImageSequenceWidget* img, uint32_t idx,
GFXU_ImageAsset* imgAst);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index
GFXU_ImageAsset* imgAst	the image asset pointer

Function

laResult laImageSequenceWidget_SetImage(laImageSequenceWidget* img,
uint32_t idx,
GFXU_ImageAsset* imgAst)

laImageSequenceWidget_SetImageChangedEventCallback Function

Sets the image changed event callback pointer. This callback is called whenever the active display index is changed.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImageChangedEventCallback(laImageSequenceWidget* img,
laImageSequenceImageChangedEvent_FnPtr cb);
```

Returns

laResult

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
laImageSequenceImageChangedEvent_FnPtr cb	a valid callback pointer or NULL

Function

laResult laImageSequenceWidget_SetImageChangedEventCallback(laImageSequenceWidget* img,
laImageSequenceImageChangedEvent_FnPtr cb)

laImageSequenceWidget_SetImageCount Function

Sets the number of image entries for this widget. An image entry that is null will show nothing.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImageCount(laImageSequenceWidget* img, uint32_t count);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t count	the desired number of entries

Function

laResult laImageSequenceWidget_SetImageCount(laImageSequenceWidget* img,
uint32_t count)

laImageSequenceWidget_SetImageDelay Function

Sets the image delay for an entry.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImageDelay(laImageSequenceWidget* img, uint32_t idx, uint32_t delay);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index
uint32_t delay	the delay value

Function

laResult laImageSequenceWidget_SetImageDelay(laImageSequenceWidget* img,
uint32_t idx,
uint32_t delay)

laImageSequenceWidget_SetImageHAlignment Function

Sets the horizontal alignment for an image entry.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImageHAlignment(laImageSequenceWidget* img, uint32_t idx, laHAlignment align);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index
laHAlignment align	the halign value

Function

laResult laImageSequenceWidget_SetImageHAlignment(laImageSequenceWidget* img,
uint32_t idx,
laHAlignment align)

laImageSequenceWidget_SetImageVAlignment Function

Sets the vertical alignment value for an image entry

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetImageVAlignment(laImageSequenceWidget* img, uint32_t idx,
laVAlignment align);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the index
laVAlignment align	the vertical alignment setting

Function

laResult laImageSequenceWidget_SetImageVAlignment(laImageSequenceWidget* img,
uint32_t idx,
laVAlignment align)

laImageSequenceWidget_SetRepeat Function

Sets the repeat flag for the widget

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_SetRepeat(laImageSequenceWidget* img, laBool repeat);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
laBool repeat	the desired repeat setting

Function

laResult laImageSequenceWidget_SetRepeat(laImageSequenceWidget* img,
laBool repeat)

laImageSequenceWidget_ShowImage Function

Sets the active display index to the indicated value.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_ShowImage(laImageSequenceWidget* img, uint32_t idx);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget
uint32_t idx	the desired index

Function

laResult laImageSequenceWidget_ShowImage(laImageSequenceWidget* img,
uint32_t idx)

laImageSequenceWidget_ShowNextImage Function

Sets the active display index to the next index value.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_ShowNextImage(laImageSequenceWidget* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laResult laImageSequenceWidget_ShowNextImage(laImageSequenceWidget* img)

laImageSequenceWidget_ShowPreviousImage Function

Sets the active display index to the previous index value.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_ShowPreviousImage(laImageSequenceWidget* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laResult laImageSequenceWidget_ShowPreviousImage(laImageSequenceWidget* img)

laImageSequenceWidget_Stop Function

Stops the widget from automatically cycling through the image entries.

File

libaria_widget_imagesequence.h

C

```
LIB_EXPORT laResult laImageSequenceWidget_Stop(laImageSequenceWidget* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laImageSequenceWidget* img	the widget

Function

laResult laImageSequenceWidget_Stop(laImageSequenceWidget* img)

laImageWidget_GetHAlignment Function

Gets the image horizontal alignment value.

File

libaria_widget_image.h

C

```
LIB_EXPORT laHAlignment laImageWidget_GetHAlignment(laImageWidget* img);
```

Returns

laHAlignment - the horizontal alignment value

Parameters

Parameters	Description
laImageWidget* img	the widget

Function

laHAlignment laImageWidget_GetHAlignment(laImageWidget* img)

laImageWidget_GetImage Function

Gets the image asset pointer for the widget.

File

libaria_widget_image.h

C

```
LIB_EXPORT GFXU_ImageAsset* laImageWidget_GetImage(laImageWidget* img);
```

Returns

GFXU_ImageAsset* - the image asset pointer

Parameters

Parameters	Description
laImageWidget* img	the widget

Function

GFXU_ImageAsset* laImageWidget_GetImage(laImageWidget* img)

laImageWidget_GetVAlignment Function

Gets the image vertical alignment value.

File

libaria_widget_image.h

C

```
LIB_EXPORT laVAlignment laImageWidget_GetVAlignment(laImageWidget* img);
```

Returns

laVAlignment - the vertical alignment setting

Parameters

Parameters	Description
laImageWidget* img	the widget

Function

laVAlignment laImageWidget_GetVAlignment(laImageWidget* img)

laImageWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_image.h

C

```
LIB_EXPORT laImageWidget* laImageWidget_New();
```

Returns

laImageWidget* - the widget

Function

laImageWidget* laImageWidget_New()

laImageWidget_SetHAlignment Function

Sets the image horizontal alignment value.

File

libaria_widget_image.h

C

```
LIB_EXPORT laResult laImageWidget_SetHAlignment(laImageWidget* img, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laImageWidget* img	the widget
laHAlignment align	the horizontal alignment value

Function

laResult laImageWidget_SetHAlignment(laImageWidget* img,

laHAlignment align)

laImageWidget_SetImage Function

Sets the image asset pointer for the widget.

File

libaria_widget_image.h

C

```
LIB_EXPORT laResult laImageWidget_SetImage(laImageWidget* img, GFXU_ImageAsset* imgAst);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
llaImageWidget* img	the widget
GFXU_ImageAsset* imgAst	the image asset pointer

Function

laResult laImageWidget_SetImage(laImageWidget* img,
GFXU_ImageAsset* imgAst)

laImageWidget_SetVAlignment Function

Sets the image vertical alignment value.

File

libaria_widget_image.h

C

```
LIB_EXPORT laResult laImageWidget_SetVAlignment(laImageWidget* img, laVAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laImageWidget* img	the widget
laVAlignment	the vertical alignment setting

Function

laResult laImageWidget_SetVAlignment(laImageWidget* img,
laVAlignment align)

laInput_GetEnabled Function

Returns the input enabled status of the current context

File

libaria_input.h

C

```
LIB_EXPORT laBool laInput_GetEnabled();
```

Returns

laBool

Description

Returns the input enabled status of the current context

Function

laBool laInput_GetEnabled()

laInput_InjectTouchDown Function

Register and track the touch down event and queue it for handling by associated widgets

File

libaria_input.h

C

```
LIB_EXPORT laResult laInput_InjectTouchDown(uint32_t id, int32_t x, int32_t y);
```

Returns

laResult

Description

Register and track the touch down event and queue it for handling by associated widgets

Function

laResult laInput_InjectTouchDown(uint32_t id, int32_t x, int32_t y)

laInput_InjectTouchMoved Function

Register and track the touch moved event and queue it for handling by associated widgets

File

libaria_input.h

C

```
LIB_EXPORT laResult laInput_InjectTouchMoved(uint32_t id, int32_t x, int32_t y);
```

Returns

laResult

Description

Register and track the touch moved event and queue it for handling by associated widgets

Function

laResult laInput_InjectTouchMoved(uint32_t id, int32_t x, int32_t y)

laInput_InjectTouchUp Function

Register and track the touch up event and queue it for handling by associated widgets

File

libaria_input.h

C

```
LIB_EXPORT laResult laInput_InjectTouchUp(uint32_t id, int32_t x, int32_t y);
```

Returns

laResult

Description

Register and track the touch up event and queue it for handling by associated widgets

Function

laResult laInput_InjectTouchUp(uint32_t id, int32_t x, int32_t y)

laInput_SetEnabled Function

Sets the input status of the current context with the specified input argument

File

libaria_input.h

C

```
LIB_EXPORT laResult laInput_SetEnabled(laBool enable);
```

Returns

laResult

Description

Sets the input status of the current context with the specified input argument

Function

laResult laInput_SetEnabled(laBool enable)

laKeyPadWidget_GetKeyAction Function

Gets the key pad cell action for a cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laKeyPadCellAction laKeyPadWidget_GetKeyAction(laKeyPadWidget* pad, uint32_t row, uint32_t col);
```

Returns

laKeyPadCellAction - the cell action value

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

laKeyPadCellAction laKeyPadWidget_GetKeyAction(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_GetKeyClickEventCallback Function

Gets the current key click event callback pointer

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laKeyPadWidget_KeyClickEvent laKeyPadWidget_GetKeyClickEventCallback(laKeyPadWidget* pad);
```

Returns

laKeyPadWidget_KeyClickEvent - the callback pointer

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget

Function

laKeyPadWidget_KeyClickEvent laKeyPadWidget_GetKeyClickEventCallback(laKeyPadWidget* pad)

laKeyPadWidget_GetKeyDrawBackground Function

Gets the background type for a key pad cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laBackgroundType laKeyPadWidget_GetKeyDrawBackground(laKeyPadWidget* pad, uint32_t row, uint32_t
col);
```

Returns

laBackgroundType - the cell background type

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

laBackgroundType laKeyPadWidget_GetKeyBackgroundType(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_GetKeyEnabled Function

Gets the enabled flag for a cell at a given row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laBool laKeyPadWidget_GetKeyEnabled(laKeyPadWidget* pad, uint32_t row, uint32_t col);
```

Returns

laBool - the flag value

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

laBool laKeyPadWidget_GetKeyEnabled(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_GetKeyImageMargin Function

Gets the key pad cell image margin value

File

libaria_widget_keypad.h

C

```
LIB_EXPORT uint32_t laKeyPadWidget_GetKeyImageMargin(laKeyPadWidget* pad, uint32_t row, uint32_t col);
```

Returns

uint32_t - the margin value

Description

The image margin value is the space between the image and the text

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

uint32_t laKeyPadWidget_GetKeyImageMargin(laKeyPadWidget* pad,

uint32_t row,

uint32_t col)

laKeyPadWidget_GetKeyImagePosition Function

Gets the image position for a key pad cell

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laRelativePosition laKeyPadWidget_GetKeyImagePosition(laKeyPadWidget* pad, uint32_t row,
uint32_t col);
```

Returns

laRelativePosition - the image position

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

laRelativePosition laKeyPadWidget_GetKeyImagePosition(laKeyPadWidget* pad,

uint32_t row,

uint32_t col)

laKeyPadWidget_GetKeyPressedImage Function

Gets the pressed icon image asset pointer for the display image for a key pad cell

File

libaria_widget_keypad.h

C

```
LIB_EXPORT GFXU_ImageAsset* laKeyPadWidget_GetKeyPressedImage(laKeyPadWidget* pad, uint32_t row, uint32_t
col);
```

Returns

GFXU_ImageAsset* - pointer to the icon image asset

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

GFXU_ImageAsset* laKeyPadWidget_GetKeyPressedImage(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_GetKeyReleasedImage Function

Gets the released icon image asset pointer for the display image for a key pad cell

File

libaria_widget_keypad.h

C

```
LIB_EXPORT GFXU_ImageAsset* laKeyPadWidget_GetKeyReleasedImage(laKeyPadWidget* pad, uint32_t row, uint32_t
col);
```

Returns

GFXU_ImageAsset* - pointer to the icon image asset

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

GFXU_ImageAsset* laKeyPadWidget_GetKeyReleasedImage(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_GetKeyText Function

Returns a copy of the display text for a given cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_GetKeyText(laKeyPadWidget* pad, uint32_t row, uint32_t col, laString*
str);
```

Returns

laResult - the result of the operation

Description

This function allocates memory for the input string argument. The application becomes responsible for the management of the memory after function completion.

The input string does not need to be initialized in any fashion before calling this function.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row

uint32_t col	the indicated column
laString* str	a pointer to an laString object

Function

laResult laKeyPadWidget_GetKeyText(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laString* str)

laKeyPadWidget_GetKeyValue Function

Gets the edit text value for a given key pad cell.

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laString* laKeyPadWidget_GetKeyValue(laKeyPadWidget* pad, uint32_t row, uint32_t col);
```

Returns

laString* - an initialized string containing a copy of the key pad cell edit value text

Description

This function allocates memory and returns a valid laString pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column

Function

laString* laKeyPadWidget_GetKeyValue(laKeyPadWidget* pad,
uint32_t row,
uint32_t col)

laKeyPadWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laKeyPadWidget* laKeyPadWidget_New(uint32_t rows, uint32_t cols);
```

Returns

laKeyPadWidget*

Parameters

Parameters	Description
uint32_t	number of rows to create number of columns to create

Function

laKeyPadWidget* laKeyPadWidget_New(uint32_t rows, uint32_t cols)

laKeyPadWidget_SetKeyAction Function

Sets the cell action type for a key pad cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyAction(laKeyPadWidget* pad, uint32_t row, uint32_t col,
laKeyPadCellAction action);
```

Returns

laResult - the result of the operation

Description

The cell action is the action that is dispatched to the Aria edit event system. This event will then be received by the active edit event receptor widget if one exists.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laKeyPadCellAction action	the desired edit action

Function

laResult laKeyPadWidget_SetKeyAction(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laKeyPadCellAction action)

laKeyPadWidget_SetKeyClickEventCallback Function

Sets the current key click event callback pointer

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyClickEventCallback(laKeyPadWidget* pad,
laKeyPadWidget_KeyClickEvent cb);
```

Returns

laResult - the result of the operation

Description

The key click event callback pointer is issued any time a button is interacted with.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
laKeyPadWidget_KeyClickEvent cb	the callback pointer

Function

laResult laKeyPadWidget_SetKeyClickEventCallback(laKeyPadWidget* pad,
laKeyPadWidget_KeyClickEvent cb)

laKeyPadWidget_SetKeyEnabled Function

Sets the enabled flag for a cell at the given row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyEnabled(laKeyPadWidget* pad, uint32_t row, uint32_t col, laBool enabled);
```

Returns

laResult - the result of the operation

Description

The enabled flag controls the visibility and interactibility of a key pad cell. This enables the key pad to be configured to match such examples as a phone dialer key pad with twelve buttons total but the buttons to the left and right of the zero button not being drawn.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laBool enabled	the flag value

Function

laResult laKeyPadWidget_SetKeyEnabled(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laBool enabled)

laKeyPadWidget_SetKeyImageMargin Function

Sets the key pad cell image margin value for a given cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyImageMargin(laKeyPadWidget* pad, uint32_t row, uint32_t col, uint32_t mg);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
uint32_t mg	the desired margin value

Function

laResult laKeyPadWidget_SetKeyImageMargin(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
uint32_t mg)

laKeyPadWidget_SetKeyImagePosition Function

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyImagePosition(laKeyPadWidget* pad, uint32_t row, uint32_t col, laRelativePosition pos);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laRelativePosition pos	the desired image position

Function

laResult laKeyPadWidget_SetKeyImagePosition(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laRelativePosition pos)

laKeyPadWidget_SetKeyPressedImage Function

Sets the pressed icon image asset pointer for a key pad cell

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyPressedImage(laKeyPadWidget* pad, uint32_t row, uint32_t col,
GFXU_ImageAsset* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
GFXU_ImageAsset* img	pointer to an image asset

Function

laResult laKeyPadWidget_SetKeyPressedImage(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
GFXU_ImageAsset* img)

laKeyPadWidget_SetKeyReleasedImage Function

Sets the released icon image asset pointer for a key pad cell

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyReleasedImage(laKeyPadWidget* pad, uint32_t row, uint32_t col,
GFXU_ImageAsset* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
GFXU_ImageAsset* img	pointer to an image asset

Function

laResult laKeyPadWidget_SetKeyReleasedImage(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
GFXU_ImageAsset* img)

laKeyPadWidget_SetKeyText Function

Sets the display text for a given cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyText(laKeyPadWidget* pad, uint32_t row, uint32_t col, laString str);
```

Returns

laResult - the result of the operation

Description

Sets the display text for a given cell at row/column

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laString str	an laString object

Function

laResult laKeyPadWidget_SetKeyText(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laString str)

laKeyPadWidget_SetKeyValue Function

Sets the edit value for a given key pad cell.

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyValue(laKeyPadWidget* pad, uint32_t row, uint32_t col, laString str);
```

Returns

laResult - the result of the operation

Description

The edit value for a key pad cell is the value that is passed to the Aria edit event management system. This may be different than the displayed

text of the cell or when the cell is using a picture icon and has no display text.

An input string that references the string table is a valid use case and the edit text will change as the active string table language changes.

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laString str	the string to set the key value to

Function

laResult laKeyPadWidget_SetKeyValue(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laString str)

laLabelWidget_GetHAlignment Function

Gets the text horizontal alignment value.

File

libaria_widget_label.h

C

```
LIB_EXPORT laHAlignment laLabelWidget_GetHAlignment(laLabelWidget* lbl);
```

Returns

laHAlignment - the horizontal alignment value

Parameters

Parameters	Description
laLabelWidget*	the widget

Function

laHAlignment laLabelWidget_GetHAlignment(laLabelWidget* lbl)

laLabelWidget_GetText Function

Gets the text value for the label.

File

libaria_widget_label.h

C

```
LIB_EXPORT laResult laLabelWidget_GetText(laLabelWidget* lbl, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laLabelWidget* lbl	the widget
laString* str	a pointer to an laString object

Function

laResult laLabelWidget_GetText(laLabelWidget* lbl, laString* str)

laLabelWidget_GetVAlignment Function

Gets the current vertical text alignment

File

libaria_widget_label.h

C

```
LIB_EXPORT laVAlignment laLabelWidget_GetVAlignment(laLabelWidget* lbl);
```

Returns

laVAlignment - the vertical alignment setting

Parameters

Parameters	Description
laLabelWidget*	the widget

Function

laVAlignment laLabelWidget_GetVAlignment(laLabelWidget* lbl)

laLabelWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_label.h

C

```
LIB_EXPORT laLabelWidget* laLabelWidget_New();
```

Returns

laLabelWidget*

Function

laLabelWidget* laLabelWidget_New()

laLabelWidget_SetHAlignment Function

Sets the text horizontal alignment value

File

libaria_widget_label.h

C

```
LIB_EXPORT laResult laLabelWidget_SetHAlignment(laLabelWidget* lbl, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laLabelWidget*	the widget
laHAlignment align	the horizontal alignment value

Function

laResult laLabelWidget_SetHAlignment(laLabelWidget* lbl,
laHAlignment align)

laLabelWidget_SetText Function

Sets the text value for the label.

File

libaria_widget_label.h

C

```
LIB_EXPORT laResult laLabelWidget_SetText(laLabelWidget* lbl, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laLabelWidget* lbl	the widget
laString str	an laString object

Function

laResult laLabelWidget_SetText(laLabelWidget* lbl, laString str)

laLabelWidget_SetVAlignment Function

Sets the vertical text alignment value

File

libaria_widget_label.h

C

```
LIB_EXPORT laResult laLabelWidget_SetVAlignment(laLabelWidget* lbl, laVAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laLabelWidget*	the widget
laVAlignment align	the vertical alignment setting

Function

laResult laLabelWidget_SetVAlignment(laLabelWidget* lbl,
laVAlignment align)

laLayer_Delete Function

Destructor for the layer object

File

libaria_layer.h

C

```
LIB_EXPORT void laLayer_Delete(laLayer* layer);
```

Returns

void

Description

Destructor for the layer object

Function

void laLayer_Delete(laLayer* layer)

laLayer_GetAlphaAmount Function

Get's the amount of alpha blending for a given layer

File

libaria_layer.h

C

```
LIB_EXPORT uint32_t laLayer_GetAlphaAmount(const laLayer* layer);
```

Returns

uint32_t - an alpha channel value from 0 - 255

Parameters

Parameters	Description
laLayer* layer	the layer

Function

uint32_t laLayer_GetAlphaAmount(const laLayer* layer)

laLayer_GetAlphaEnable Function

Gets the layer alpha enable flag

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetAlphaEnable(const laLayer* layer);
```

Returns

laBool - the value of the alpha enable flag

Parameters

Parameters	Description
const laLayer*	the layer

Function

laBool laLayer_GetAlphaEnable(const laLayer* layer)

laLayer_GetBufferCount Function

Return the buffer count for the current layer

File

libaria_layer.h

C

```
LIB_EXPORT uint32_t laLayer_GetBufferCount(const laLayer* layer);
```

Returns

uint32_t - the current number of buffers for the layer

Description

Return the buffer count for the current layer

Parameters

Parameters	Description
laLayer* layer	the layer

Function

uint32_t laLayer_GetBufferCount(const laLayer* layer)

laLayer_GetMaskColor Function

Returns the mask color value for the current layer

File

libaria_layer.h

C

```
LIB_EXPORT GFX_Color laLayer_GetMaskColor(const laLayer* layer);
```

Returns

GFX_Color - the layer mask color value

Description

Returns the mask color value for the current layer

Parameters

Parameters	Description
laLayer* layer	the layer

Function

GFX_Color laLayer_GetMaskColor(const laLayer* layer)

laLayer_GetMaskEnable Function

Gets the layer mask enable flag

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetMaskEnable(const laLayer* layer);
```

Returns

laBool - the value of the mask enable flag

Description

Gets the layer mask enable flag

Parameters

Parameters	Description
laLayer* layer	the layer

Function

laBool laLayer_GetMaskEnable(const laLayer* layer)

laLayer_GetVSync Function

Gets the layer's vsync flag setting

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetVSync(const laLayer* layer);
```

Returns

laBool - the state of the layer's vsync flag

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

laBool laLayer_GetVSync(const laLayer* layer)

laLayer_New Function

Constructor for a new layer

File

libaria_layer.h

C

```
LIB_EXPORT laLayer* laLayer_New();
```

Returns

laLayer*

Description

Constructor for a new layer, returns the layer object

Remarks

Allocates memory for a layer using the active context memory interface. Once added to a screen the it becomes the responsibility of the framework to free the memory.

Function

laLayer* laLayer_New()

laLayer_SetAlphaAmount Function

Set's the amount of alpha blending for a given layer

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetAlphaAmount(laLayer* layer, uint32_t amount);
```

Returns

laResult - success if the operation succeeded

Description

Set's the amount of alpha blending for a given layer

Parameters

Parameters	Description
laLayer* layer	the layer
uint32_t amount	an alpha amount from 0 - 255

Function

laResult laLayer_SetAlphaAmount(laLayer* layer, uint32_t amount)

laLayer_SetAlphaEnable Function

Sets the layer alpha enable flag to the specified value

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetAlphaEnable(laLayer* layer, laBool enable);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLayer* layer	the layer
laBool enable	the desired value of the flag

Function

laResult laLayer_SetAlphaEnable(laLayer* layer, laBool enable)

laLayer_SetBufferCount Function

Set the buffer count for the current layer to the specified value

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetBufferCount(laLayer* layer, uint32_t count);
```

Returns

laResult - the result of the operation

Description

Set the buffer count for the current layer to the specified value

Parameters

Parameters	Description
laLayer* layer	the layer
uint32_t count	the desired number of buffers

Function

laResult laLayer_SetBufferCount(laLayer* layer, uint32_t count)

laLayer_SetMaskColor Function

Set the mask color value for the current layer to the specified value

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetMaskColor(laLayer* layer, GFX_Color color);
```

Returns

laResult - the result of the operation

Description

Set the mask color value for the current layer to the specified value

Parameters

Parameters	Description
laLayer* layer	the layer
GFX_color color	the desired mask color value

Function

void laLayer_SetMaskColor(laLayer* layer, GFX_Color color)

laLayer_SetMaskEnable Function

Sets the layer mask enable flag to the specified value

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetMaskEnable(laLayer* layer, laBool enable);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLayer* layer	the layer
laBool enable	the desired value of the flag

Function

laResult laLayer_SetMaskEnable(laLayer* layer, laBool enable)

laLayer_SetVSync Function

Sets the layer's vsync flag.

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetVSync(laLayer* layer, laBool enable);
```

Returns

laResult - the result of the operation

Description

Sets the layer's vsync flag.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

void laLayer_SetVSync(laLayer* layer, laBool enable)

laLineWidget_GetEndPoint Function

Gets the coordinates for the second point of the line.

File

libaria_widget_line.h

C

```
LIB_EXPORT laResult laLineWidget_GetEndPoint(laLineWidget* line, int32_t* x, int32_t* y);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLineWidget* line	the widget
int32_t* x	pointer to an int to store the x coordinate
int32_t* y	pointer to tan int to store the y coordinate

Function

laResult laLineWidget_GetEndPoint(laLineWidget* line, int32_t* x, int32_t* y)

laLineWidget_GetStartPoint Function

Gets the coordinates for the first point of the line.

File

libaria_widget_line.h

C

```
LIB_EXPORT laResult laLineWidget_GetStartPoint(laLineWidget* line, int32_t* x, int32_t* y);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLineWidget* line	the widget
int32_t* x	pointer to an int to store the x coordinate
int32_t* y	pointer to tan int to store the y coordinate

Function

laResult laLineWidget_GetStartPoint(laLineWidget* line, int32_t* x, int32_t* y)

laLineWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_line.h

C

```
LIB_EXPORT laLineWidget* laLineWidget_New();
```

Returns

laLineWidget*

Function

laLineWidget* laLineWidget_New()

laLineWidget_SetEndPoint Function

Sets the cooridnate for the second point of the line

File

libaria_widget_line.h

C

```
LIB_EXPORT laResult laLineWidget_SetEndPoint(laLineWidget* line, int32_t x, int32_t y);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLineWidget* line	the widget
int32_t x	the x coordinate value
int32_t y	the y coordinate value

Function

laResult laLineWidget_SetEndPoint(laLineWidget* line, int32_t x, int32_t y)

laLineWidget_SetStartPoint Function

Sets the cooridnate for the first point of the line

File

libaria_widget_line.h

C

```
LIB_EXPORT laResult laLineWidget_SetStartPoint(laLineWidget* line, int32_t x, int32_t y);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLineWidget* line	the widget
int32_t x	the x coordinate value
int32_t y	the y coordinate value

Function

laResult laLineWidget_SetStartPoint(laLineWidget* line, int32_t x, int32_t y)

laListWheelWidget_AppendItem Function

Appends a new item entry to the list. The initial value of the item will be empty.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_AppendItem(laListWheelWidget* whl);
```

Returns

uint32_t - the index of the newly appended item

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_AppendItem(laListWheelWidget* whl)

laListWheelWidget_GetAlignment Function

Gets the horizontal alignment for the list widget

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laHAlignment laListWheelWidget_GetAlignment(laListWheelWidget* whl);
```

Returns

laHAlignment - the current list halign mode

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laHAlignment laListWheelWidget_GetAlignment(laListWheelWidget* whl)

laListWheelWidget_GetIconMargin Function

Gets the icon margin value for the list wheel widget

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetIconMargin(laListWheelWidget* whl);
```

Returns

uint32_t - the icon margin value

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetIconMargin(laListWheelWidget* whl)

laListWheelWidget_GetIconPosition Function

Sets the icon position for the list wheel widget.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laRelativePosition laListWheelWidget_GetIconPosition(laListWheelWidget* whl);
```

Returns

laRelativePosition - the current icon position

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laRelativePosition laListWheelWidget_GetIconPosition(laListWheelWidget* whl)

laListWheelWidget_GetItemCount Function

Gets the number of items currently contained in the list

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetItemCount(laListWheelWidget* whl);
```

Returns

uint32_t - the number of items in the list

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetItemCount(laListWheelWidget* whl)

laListWheelWidget_GetItemIcon Function

Gets the pointer to the image asset for the icon for the item at the given index.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT GFXU_ImageAsset* laListWheelWidget_GetItemIcon(laListWheelWidget* whl, uint32_t index);
```

Returns

GFXU_ImageAsset* - the image asset pointer or NULL

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the index to consider

Function

GFXU_ImageAsset* laListWheelWidget_GetItemIcon(laListWheelWidget* whl,
uint32_t index)

laListWheelWidget_GetItemText Function

Gets the text value for an item in the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_GetItemText(laListWheelWidget* whl, uint32_t idx, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the index to consider
laString* str	a pointer to an laString object

Function

laResult laListWheelWidget_GetItemText(laListWheelWidget* whl,
uint32_t idx,
laString* str)

laListWheelWidget_GetSelectedItem Function

Returns the index of the currently selected item.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT int32_t laListWheelWidget_GetSelectedItem(laListWheelWidget* whl);
```

Returns

int32_t - the index of the selected item or -1 if an error occured

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

int32_t laListWheelWidget_GetSelectedItem(laListWheelWidget* whl)

laListWheelWidget_GetSelectedItemChangedEventCallback Function

Gets the callback for the item selected changed event

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laListWheelWidget_SelectedItemChangedEvent
laListWheelWidget_GetSelectedItemChangedEventCallback(laListWheelWidget* whl);
```

Returns

laListWheelWidget_SelectedItemChangedEvent - the current pointer to the callback or NULL

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laListWheelWidget_SelectedItemChangedEvent laListWheelWidget_GetSelectedItemChangedEventCallback(laListWheelWidget* whl)

laListWheelWidget_InsertItem Function

Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_InsertItem(laListWheelWidget* whl, uint32_t idx);
```

Returns

uint32_t - the index of the inserted item

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the desired index of the new item

Function

uint32_t laListWheelWidget_InsertItem(laListWheelWidget* whl, uint32_t idx)

laListWheelWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laListWheelWidget* laListWheelWidget_New();
```

Returns

laListWheelWidget*

Function

laListWheelWidget* laListWheelWidget_New()

laListWheelWidget_RemoveAllItems Function

Attempts to remove all items from the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_RemoveAllItems(laListWheelWidget* whl);
```

Returns

laResult - the operation result

Remarks

All memory owned by each item string will be freed automatically.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laResult laListWheelWidget_RemoveAllItems(laListWheelWidget* whl)

laListWheelWidget_RemoveItem Function

Attempts to remove an item from the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_RemoveItem(laListWheelWidget* whl, uint32_t idx);
```

Returns

laResult - the operation result

Remarks

The memory owned by the string item will be freed automatically.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the index to remove from the list

Function

laResult laListWheelWidget_RemoveItem(laListWheelWidget* whl, uint32_t idx)

laListWheelWidget_SelectNextItem Function

Attempts to move the selected item index to the next item in the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SelectNextItem(laListWheelWidget* whl);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laResult laListWheelWidget_SelectNextItem(laListWheelWidget* whl)

laListWheelWidget_SelectPreviousItem Function

Attempts to move the selected item index to the previous item in the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SelectPreviousItem(laListWheelWidget* whl);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laResult laListWheelWidget_SelectPreviousItem(laListWheelWidget* whl)

laListWheelWidget_SetAlignment Function

Sets the horizontal alignment mode for the list widget.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetAlignment(laListWheelWidget* whl, laHAlignment align);
```

Returns

laResult

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
laHAlignment align	the desired halign mode

Function

laResult laListWheelWidget_SetAlignment(laListWheelWidget* whl,
laHAlignment align)

laListWheelWidget_SetIconMargin Function

Sets the icon margin value for the list widget.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetIconMargin(laListWheelWidget* whl, uint32_t mg);
```

Returns

laResult - the operation result

Description

The icon margin value is the distance between the icon image and the text.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t mg	the margin value

Function

laResult laListWheelWidget_SetIconMargin(laListWheelWidget* whl, uint32_t mg)

laListWheelWidget_SetIconPosition Function

Sets the icon position for the list wheel widget

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetIconPosition(laListWheelWidget* whl, laRelativePosition pos);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
laRelativePosition pos	the relative position setting

Function

laResult laListWheelWidget_SetIconPosition(laListWheelWidget* whl,
laRelativePosition pos)

laListWheelWidget_SetItemIcon Function

Sets the icon pointer for a given index.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetItemIcon(laListWheelWidget* whl, uint32_t index, GFXU_ImageAsset*
img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the index to configure
GFXU_ImageAsset*	the image asset pointer to use as the icon

Function

laResult laListWheelWidget_SetItemIcon(laListWheelWidget* whl,
uint32_t index,
GFXU_ImageAsset* img)

laListWheelWidget_SetItemText Function

Sets the text value for an item in the list.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetItemText(laListWheelWidget* whl, uint32_t index, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the index to consider
laString str	an laString object

Function

laResult laListWheelWidget_SetItemText(laListWheelWidget* whl,
uint32_t index,
laString str)

laListWheelWidget_SetSelectedItem Function

Attempts to set the selectedi tem index

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetSelectedItem(laListWheelWidget* whl, uint32_t idx);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t idx	the desired selected item index

Function

laResult laListWheelWidget_SetSelectedItem(laListWheelWidget* whl,
uint32_t idx)

laListWheelWidget_SetSelectedItemChangedEventCallback Function

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetSelectedItemChangedEventCallback(laListWheelWidget* whl,
laListWheelWidget_SelectedItemChangedEvent cb);
```

Returns

laResult - the operation result

Description

This callback is called whenver the wheel's selected item changes.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
laListWheelWidget_SelectedItemChangedEvent	the desired pointer to callback or NULL

Function

laResult laListWheelWidget_SetSelectedItemChangedEventCallback(laListWheelWidget* whl,
laListWheelWidget_SelectedItemChangedEvent cb)

laListWidget_AppendItem Function

Appends a new item entry to the list. The initial value of the item will be empty.

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_AppendItem(laListWidget* lst);
```

Returns

uint32_t - the index of the newly appended item

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_AppendItem(laListWidget* lst)

laListWidget_DeselectAll Function

Attempts to set all item states as not selected.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_DeselectAll(laListWidget* lst);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laResult laListWidget_DeselectAll(laListWidget* lst)

laListWidget_GetAlignment Function

Gets the horizontal alignment for the list widget

File

libaria_widget_list.h

C

```
LIB_EXPORT laHAlignment laListWidget_GetAlignment(laListWidget* lst);
```

Returns

laHAlignment - the current list halign mode

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laHAlignment laListWidget_GetAlignment(laListWidget* lst)

laListWidget_GetAllowEmptySelection Function

Returns true of the list allows an empty selection set

File

libaria_widget_list.h

C

```
LIB_EXPORT laBool laListWidget_GetAllowEmptySelection(laListWidget* lst);
```

Returns

laBool - true if the list allows an empty selection set

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laBool laListWidget_GetAllowEmptySelection(laListWidget* lst)

laListWidget_GetFirstSelectedItem Function

Returns the lowest selected item index.

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_GetFirstSelectedItem(laListWidget* lst);
```

Returns

uint32_t - the lowest selected item index or -1 if nothing is selected.

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_GetFirstSelectedItem(laListWidget* lst)

laListWidget_GetIconMargin Function

Gets the icon margin value for the list widget

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_GetIconMargin(laListWidget* lst);
```

Returns

uint32_t - the icon margin value

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_GetIconMargin(laListWidget* lst)

laListWidget_GetIconPosition Function

Gets the icon position for the list

File

libaria_widget_list.h

C

```
LIB_EXPORT laRelativePosition laListWidget_GetIconPosition(laListWidget* lst);
```

Returns

laRelativePosition - the current icon position

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laRelativePosition laListWidget_GetIconPosition(laListWidget* lst)

laListWidget_GetItemCount Function

Gets the number of items currently contained in the list

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_GetItemCount(laListWidget* lst);
```

Returns

uint32_t - the number of items in the list

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_GetItemCount(laListWidget* lst)

laListWidget_GetItemIcon Function

Gets the pointer to the image asset for the icon for the item at the given index.

File

libaria_widget_list.h

C

```
LIB_EXPORT GFXU_ImageAsset* laListWidget_GetItemIcon(laListWidget* lst, uint32_t idx);
```

Returns

GFXU_ImageAsset* - the image asset pointer or NULL

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider

Function

GFXU_ImageAsset* laListWidget_GetItemIcon(laListWidget* lst,
uint32_t idx)

laListWidget_GetItemSelected Function

Returns true if the item at the given index is currently selected.

File

libaria_widget_list.h

C

```
LIB_EXPORT laBool laListWidget_GetItemSelected(laListWidget* lst, uint32_t idx);
```

Returns

laBool - the selection state of the item

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider

Function

laBool laListWidget_GetItemSelected(laListWidget* lst,
uint32_t idx)

laListWidget_GetItemText Function

Gets the text value for an item in the list.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_GetItemText(laListWidget* lst, uint32_t idx, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider
laString* str	a pointer to an laString object

Function

laResult laListWidget_GetItemText(laListWidget* lst,
uint32_t idx,
laString* str)

laListWidget_GetLastSelectedItem Function

Returns the highest selected item index.

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_GetLastSelectedItem(laListWidget* lst);
```

Returns

uint32_t - the highest selected item index or -1 if nothing is selected.

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_GetLastSelectedItem(laListWidget* lst)

laListWidget_GetSelectedItemChangedEventCallback Function

Gets the callback for the item selected changed event

File

libaria_widget_list.h

C

```
LIB_EXPORT laListWidget_SelectedItemChangedEvent
```

```
laListWidget_GetSelectedItemChangedEventCallback(laListWidget* lst);
```

Returns

laListWidget_SelectedItemChangedEvent - the current pointer to callback or NULL

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laListWidget_SelectedItemChangedEvent laListWidget_GetSelectedItemChangedEventCallback(laListWidget* lst)

laListWidget_GetSelectionCount Function

Returns the number of selected items in the list.

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_GetSelectionCount(laListWidget* lst);
```

Returns

uint32_t - the number of selected items

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWidget_GetSelectionCount(laListWidget* lst)

laListWidget_GetSelectionMode Function

Gets the selection mode for the list

File

libaria_widget_list.h

C

```
LIB_EXPORT laListWidget_SelectionMode laListWidget_GetSelectionMode(laListWidget* lst);
```

Returns

laListWidget_SelectionMode - the list selection mode

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laListWidget_SelectionMode laListWidget_GetSelectionMode(laListWidget* lst)

laListWidget_InsertItem Function

Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.

File

libaria_widget_list.h

C

```
LIB_EXPORT uint32_t laListWidget_InsertItem(laListWidget* lst, uint32_t idx);
```

Returns

uint32_t - the index of the inserted item

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the desired index of the new item

Function

uint32_t laListWidget_InsertItem(laListWidget* lst, uint32_t idx)

laListWidget_New Function

Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_list.h

C

```
LIB_EXPORT laListWidget* laListWidget_New();
```

Returns

laListWidget* lst - the widget

Function

laListWidget* laListWidget_New()

laListWidget_RemoveAllItems Function

Attempts to remove all items from the list.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_RemoveAllItems(laListWidget* lst);
```

Returns

laResult - the operation result

Remarks

All memory owned by each item string will be freed automatically.

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laResult laListWidget_RemoveAllItems(laListWidget* lst)

laListWidget_RemoveItem Function

Attempts to remove an item from the list.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_RemoveItem(laListWidget* lst, uint32_t idx);
```

Returns

laResult - the operation result

Remarks

The memory owned by the string item will be freed automatically.

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to remove from the list

Function

laResult laListWidget_RemoveItem(laListWidget* lst, uint32_t idx)

laListWidget_SelectAll Function

Attempts to set all item states to selected.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SelectAll(laListWidget* lst);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

laResult laListWidget_SelectAll(laListWidget* lst)

laListWidget_SetAlignment Function

Sets the horizontal alignment mode for the list widget.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetAlignment(laListWidget* lst, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
laHAlignment align	the desired halign mode

Function

laResult laListWidget_SetAlignment(laListWidget* lst,
laHAlignment align)

laListWidget_SetAllowEmptySelection Function

Configures the list to allow an empty selection set.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetAllowEmptySelection(laListWidget* lst, laBool allow);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
laBool allow	the desired empty selection set mode

Function

laResult laListWidget_SetAllowEmptySelection(laListWidget* lst,
laBool allow)

laListWidget_SetIconMargin Function

Sets the icon margin value for the list widget.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetIconMargin(laListWidget* lst, uint32_t mg);
```

Returns

laResult - the operation result

Description

The icon margin value is the distance between the icon image and the text.

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t mg	the margin value

Function

laResult laListWidget_SetIconMargin(laListWidget* lst, uint32_t mg)

laListWidget_SetIconPosition Function

Sets the icon position for the list widget

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetIconPosition(laListWidget* lst, laRelativePosition pos);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
laRelativePosition pos	the relative position setting

Function

laResult laListWidget_SetIconPosition(laListWidget* lst,
laRelativePosition pos)

laListWidget_SetItemIcon Function

Sets the icon pointer for a given index.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetItemIcon(laListWidget* lst, uint32_t idx, GFXU_ImageAsset* img);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to configure
GFXU_ImageAsset*	the image asset pointer to use as the icon

Function

laResult laListWidget_SetItemIcon(laListWidget* lst,
uint32_t idx,
GFXU_ImageAsset* img)

laListWidget_SetItemSelected Function

Attempts to set the item at idx as selected.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetItemSelected(laListWidget* lst, uint32_t idx, laBool selected);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider
laBool	the select state to set to the item

Function

laResult laListWidget_SetItemSelected(laListWidget* lst,
uint32_t idx,
laBool selected)

laListWidget_SetItemText Function

Sets the text value for an item in the list.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetItemText(laListWidget* lst, uint32_t index, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider
laString str	an laString object

Function

laResult laListWidget_SetItemText(laListWidget* lst,
uint32_t index,
laString str)

laListWidget_SetItemVisible Function

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetItemVisible(laListWidget* lst, uint32_t idx);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t index	the index to modify

Function

laResult laListWidget_SetItemVisible(laListWidget* lst,
uint32_t index)

laListWidget_SetSelectedItemChangedEventCallback Function

Sets the callback for the item selected changed event

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetSelectedItemChangedEventCallback(laListWidget* lst,
laListWidget_SelectedItemChangedEvent cb);
```

Returns

laResult - the operation result

Description

This callback is called whenver an items selected state is modified.

Parameters

Parameters	Description
laListWidget* lst	the widget
laListWidget_SelectedItemChangedEvent	the desired pointer to callback or NULL

Function

laResult laListWidget_SetSelectedItemChangedEventCallback(laListWidget* lst,
laListWidget_SelectedItemChangedEvent cb)

laListWidget_SetSelectionMode Function

Set the list selection mode

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_SetSelectionMode(laListWidget* lst, laListWidget_SelectionMode mode);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
laListWidget_SelectionMode mode	the desired list selection mode

Function

laResult laListWidget_SetSelectionMode(laListWidget* lst,
laListWidget_SelectionMode mode)

laListWidget_ToggleItemSelected Function

Attempts to toggle the selected state of the item at idx.

File

libaria_widget_list.h

C

```
LIB_EXPORT laResult laListWidget_ToggleItemSelected(laListWidget* lst, uint32_t idx);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t idx	the index to consider

Function

laResult laListWidget_ToggleItemSelected(laListWidget* lst,
uint32_t idx)

laProgressBarWidget_GetDirection Function

Gets the fill direction value for a progress bar widget

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laProgressBarDirection laProgressBarWidget_GetDirection(laProgressBarWidget* bar);
```

Returns

laProgressBarDirection - the fill direction value

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget

Function

laProgressBarDirection laProgressBarWidget_GetDirection(laProgressBarWidget* bar)

laProgressBarWidget_GetValue Function

Gets the percentage value for a progress bar.

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT uint32_t laProgressBarWidget_GetValue(laProgressBarWidget* bar);
```

Returns

uint32_t

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget

Function

uint32_t laProgressBarWidget_GetValue(laProgressBarWidget* bar)

laProgressBarWidget_GetValueChangedEventCallback Function

Gets the currently set value changed event callback.

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laProgressBar_ValueChangedEventCallback
laProgressBarWidget_GetValueChangedEventCallback(laProgressBarWidget* bar);
```

Returns

laProgressBar_ValueChangedEventCallback - the current callback pointer or NULL

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget

Function

laProgressBar_ValueChangedEventCallback laProgressBarWidget_GetValueChangedEventCallback(laProgressBarWidget* bar)

laProgressBarWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laProgressBarWidget* laProgressBarWidget_New();
```

Returns

laProgressBarWidget*

Function

laProgressBarWidget* laProgressBarWidget_New()

laProgressBarWidget_SetDirection Function

Sets the fill direction for a progress bar widget

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laResult laProgressBarWidget_SetDirection(laProgressBarWidget* bar, laProgressBarDirection dir);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget
laProgressBarDirection dir	the desired fill direction

Function

laResult laProgressBarWidget_SetDirection(laProgressBarWidget* bar,
laProgressBarDirection dir)

laProgressBarWidget_SetValue Function

Sets the percentage value for a progress bar. Valid values are 0 - 100.

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laResult laProgressBarWidget_SetValue(laProgressBarWidget* bar, uint32_t value);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget
uint32_t value	the desired value

Function

laResult laProgressBarWidget_SetValue(laProgressBarWidget* bar, uint32_t value)

laProgressBarWidget_SetValueChangedCallback Function

Sets the desired value changed event callback pointer

File

libaria_widget_progressbar.h

C

```
LIB_EXPORT laResult laProgressBarWidget_SetValueChangedCallback(laProgressBarWidget* bar,
laProgressBar_ValueChangedEventCallback cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laProgressBarWidget* bar	the widget
laProgressBar_ValueChangedEventCallback	a valid callback pointer or NULL

Function

laResult laProgressBarWidget_SetValueChangedCallback(laProgressBarWidget* bar, laProgressBar_ValueChangedEventCallback cb)

laRadioButtonGroup_AddButton Function

Add a button widget to the button list of the selected Radio button group.

File

libaria_radiobutton_group.h

C

```
LIB_EXPORT laResult laRadioButtonGroup_AddButton(laRadioButtonGroup* grp, laRadioButtonWidget* btn);
```

Returns

laResult

Description

Add a button widget to the button list of the selected Radio button group. The function makes sure the radio button grp is valid and the button widget to be added is not already a part of the group. The button is then added as the last button in the group button list

Function

laResult laRadioButtonGroup_AddButton(laRadioButtonGroup* grp,
laRadioButtonWidget* btn)

laRadioButtonGroup_Create Function

This function creates a GFX_GOL_RADIOBUTTON group with the provided button list.

File

libaria_radiobutton_group.h

C

```
LIB_EXPORT laResult laRadioButtonGroup_Create(laRadioButtonGroup** grp);
```

Returns

laResult

Description

This function creates a GFX_GOL_RADIOBUTTON group with the given pointer and the button list provided within the laRadioButtonGroup object.

Function

laResult laRadioButtonGroup_Create(laRadioButtonGroup** grp)

laRadioButtonGroup_Destroy Function

This function destroys the GFX_GOL_RADIOBUTTON group

File

libaria_radiobutton_group.h

C

```
LIB_EXPORT void laRadioButtonGroup_Destroy(laRadioButtonGroup* grp);
```

Returns

void

Description

This function destroys he GFX_GOL_RADIOBUTTON group with the given pointer. It frees the memory allocated to the button group and clears the button list.

Function

void laRadioButtonGroup_Destroy(laRadioButtonGroup* grp)

laRadioButtonGroup_RemoveButton Function

Remove a button widget to the button list of the selected Radio button group.

File

libaria_radiobutton_group.h

C

```
LIB_EXPORT laResult laRadioButtonGroup_RemoveButton(laRadioButtonGroup* grp, laRadioButtonWidget* btn);
```

Returns

laResult

Description

Remove a button widget to the button list of the selected Radio button group. The function makes sure the radio button grp is valid and the button widget to be removed is a part of the group. The button is then removed properly making sure to handle the list correctly. If the list size is 0, the group is destroyed.

Function

laResult laRadioButtonGroup_RemoveButton(laRadioButtonGroup* grp,
laRadioButtonWidget* btn);

laRadioButtonGroup_SelectButton Function

Select the button widget specified from the button list for the Radio button group.

File

libaria_radiobutton_group.h

C

```
LIB_EXPORT laResult laRadioButtonGroup_SelectButton(laRadioButtonGroup* grp, laRadioButtonWidget* btn);
```

Returns

laResult

Description

Select the button widget specified from the button list for the Radio button group. The function makes sure the specified button widget is a part of the group. It deselects the currently selected button widget and reassigns the focus to the button widget specified.

Function

laResult laRadioButtonGroup_SelectButton(laRadioButtonGroup* grp,
laRadioButtonWidget* btn)

laRadioButtonWidget_GetDeselectedEventCallback Function

Gets the current radio button deselected event callback

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laRadioButtonWidget_DeselectedEvent
laRadioButtonWidget_GetDeselectedEventCallback(laRadioButtonWidget* btn);
```

Returns

laRadioButtonWidget_DeselectedEvent - a valid callback pointer or NULL

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laRadioButtonWidget_DeselectedEvent laRadioButtonWidget_GetDeselectedEventCallback(laRadioButtonWidget* btn)

laRadioButtonWidget_GetGroup Function

Returns the pointer to the currently set radio button group.

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laRadioButtonGroup* laRadioButtonWidget_GetGroup(laRadioButtonWidget* btn);
```

Returns

laRadioButtonGroup* - the currently assigned radio button group

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laRadioButtonGroup* laRadioButtonWidget_GetGroup(laRadioButtonWidget* btn)

laRadioButtonWidget_GetHAlignment Function

Gets the horizontal alignment setting for a button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laHAlignment laRadioButtonWidget_GetHAlignment(laRadioButtonWidget* btn);
```

Returns

laHAlignment - the horizontal alignment value

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laHAlignment laRadioButtonWidget_GetHAlignment(laRadioButtonWidget* btn)

laRadioButtonWidget_GetImageMargin Function

Gets the distance between the icon and the text

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT uint32_t laRadioButtonWidget_GetImageMargin(laRadioButtonWidget* btn);
```

Returns

uint32_t - the distance value

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

uint16_t laRadioButtonWidget_GetImageMargin(laRadioButtonWidget* btn)

laRadioButtonWidget_GetImagePosition Function

Gets the current image position setting for the radio button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laRelativePosition laRadioButtonWidget_GetImagePosition(laRadioButtonWidget* btn);
```

Returns

laRelativePosition - the current image relative position

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laRelativePosition laRadioButtonWidget_GetImagePosition(laRadioButtonWidget* btn)

laRadioButtonWidget_GetSelected Function

Returns true if this radio button is currently selected

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laBool laRadioButtonWidget_GetSelected(laRadioButtonWidget* btn);
```

Returns

laBool - true if this button is currently selected

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laBool laRadioButtonWidget_GetSelected(laRadioButtonWidget* btn)

laRadioButtonWidget_GetSelectedEventCallback Function

Gets the current radio button selected event callback

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laRadioButtonWidget_SelectedEvent
laRadioButtonWidget_GetSelectedEventCallback(laRadioButtonWidget* btn);
```

Returns

laRadioButtonWidget_SelectedEvent - a valid callback pointer or NULL

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laRadioButtonWidget_SelectedEvent laRadioButtonWidget_GetSelectedEventCallback(laRadioButtonWidget* btn)

laRadioButtonWidget_GetSelectedImage Function

Gets the selected image asset pointer for a button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT GFXU_ImageAsset* laRadioButtonWidget_GetSelectedImage(laRadioButtonWidget* btn);
```

Returns

GFXU_ImageAsset* - the selected asset pointer

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

GFXU_ImageAsset* laRadioButtonWidget_GetSelectedImage(laRadioButtonWidget* btn)

laRadioButtonWidget_GetText Function

Gets the text value for the button.

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_GetText(laRadioButtonWidget* btn, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laString* str	a pointer to an laString object

Function

laResult laRadioButtonWidget_GetText(laRadioButtonWidget* btn,
laString* str)

laRadioButtonWidget_GetUnselectedImage Function

Gets the image asset pointer currently used as the unselected icon

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT GFXU_ImageAsset* laRadioButtonWidget_GetUnselectedImage(laRadioButtonWidget* btn);
```

Returns

GFXU_ImageAsset* - the selected asset pointer

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

GFXU_ImageAsset* laRadioButtonWidget_GetUnselectedImage(laRadioButtonWidget* btn)

laRadioButtonWidget_GetVAlignment Function

Sets the vertical alignment for a button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laVAlignment laRadioButtonWidget_GetVAlignment(laRadioButtonWidget* btn);
```

Returns

laVAlignment align - the desired vertical alignment setting

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laVAlignment laRadioButtonWidget_GetVAlignment(laRadioButtonWidget* btn)

laRadioButtonWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laRadioButtonWidget* laRadioButtonWidget_New();
```

Returns

laProgressBarWidget*

Function

laRadioButtonWidget* laRadioButtonWidget_New()

laRadioButtonWidget_SetDeselectedEventCallback Function

Sets the deselected callback pointer

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetDeselectedEventCallback(laRadioButtonWidget* btn,
laRadioButtonWidget_DeselectedEvent cb);
```

Returns

laResult - the operation result

Description

This callback is called when this radio button is deselected

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laRadioButtonWidget_DeselectedEvent	a valid callback pointer or NULL

Function

laResult laRadioButtonWidget_SetDeselectedEventCallback(laRadioButtonWidget* btn,
laRadioButtonWidget_DeselectedEvent cb)

laRadioButtonWidget_SetHAlignment Function

Sets the horizontal alignment value for a button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetHAlignment(laRadioButtonWidget* btn, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laHAlignment align	the desired alignment value

Function

laResult laRadioButtonWidget_SetHAlignment(laRadioButtonWidget* btn,
laHAlignment align)

laRadioButtonWidget_SetImagePosition Function

Sets the image relative position setting for the radio button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetImagePosition(laRadioButtonWidget* btn, laRelativePosition pos);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laRelativePosition pos	the desired image position

Function

laResult laRadioButtonWidget_SetImagePosition(laRadioButtonWidget* btn,
laRelativePosition pos)

laRadioButtonWidget_SetSelected Function

Sets this button as selected.

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetSelected(laRadioButtonWidget* btn);
```

Returns

laResult - the operation result

Description

If this button belongs to a radio button group then this function will potentially unselect another button and become selected.

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget

Function

laResult laRadioButtonWidget_SetSelected(laRadioButtonWidget* btn)

laRadioButtonWidget_SetSelectedEventCallback Function

Sets the radio button selected event callback

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetSelectedEventCallback(laRadioButtonWidget* btn,
laRadioButtonWidget_SelectedEvent cb);
```

Returns

laResult - the operation result

Description

This callback is called when the radio button becomes selected

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laRadioButtonWidget_SelectedEvent	a valid callback pointer or NULL

Function

laResult laRadioButtonWidget_SetSelectedEventCallback(laRadioButtonWidget* btn,
laRadioButtonWidget_SelectedEvent cb)

laRadioButtonWidget_SetSelectedImage Function

Sets the image to be used as a selected icon

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetSelectedImage(laRadioButtonWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
GFXU_ImageAsset* img	the desired image asset pointer or NULL

Function

laResult laRadioButtonWidget_SetSelectedImage(laRadioButtonWidget* btn,
GFXU_ImageAsset* img)

laRadioButtonWidget_SetText Function

Sets the text value for the button.

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetText(laRadioButtonWidget* btn, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laString str	an laString object

Function

laResult laRadioButtonWidget_SetText(laRadioButtonWidget* btn,
laString str)

laRadioButtonWidget_SetUnselectedImage Function

Sets the asset pointer for the radio button's unselected image icon

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetUnselectedImage(laRadioButtonWidget* btn, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
GFXU_ImageAsset* img	the desired image asset pointer or NULL

Function

laResult laRadioButtonWidget_SetUnselectedImage(laRadioButtonWidget* btn,
GFXU_ImageAsset* img)

laRadioButtonWidget_SetVAlignment Function

Sets the vertical alignment for a button

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetVAlignment(laRadioButtonWidget* btn, laVAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
laVAlignment align	the desired vertical alignment setting

Function

laResult laRadioButtonWidget_SetVAlignment(laRadioButtonWidget* btn,
laVAlignment align)

laRectangleWidget_GetThickness Function

Gets the rectangle border thickness setting

File

libaria_widget_rectangle.h

C

```
LIB_EXPORT int32_t laRectangleWidget_GetThickness(laRectangleWidget* rect);
```

Returns

int32_t - the border thickness setting

Parameters

Parameters	Description
laRectangleWidget* rect	the widget

Function

int32_t laRectangleWidget_GetThickness(laRectangleWidget* rect)

laRectangleWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_rectangle.h

C

```
LIB_EXPORT laRectangleWidget* laRectangleWidget_New();
```

Returns

laRectangleWidget*

Function

laRectangleWidget* laRectangleWidget_New()

laScheme_Initialize Function

Initialize the scheme to the default values as per the specified color mode.

File

libaria_scheme.h

C

```
LIB_EXPORT void laScheme_Initialize(laScheme* scheme, GFX_ColorMode mode);
```

Returns

void

Description

Initialize the scheme to the default values as per the specified color mode.

Parameters

Parameters	Description
laSceme* scheme	the scheme to modify
GFX_ColorMode	the color mode to use

Function

void laScheme_Initialize(laScheme* scheme, GFX_ColorMode mode)

laScreen_Delete Function

Frees all memory for all layers and widgets for this screen

File

libaria_screen.h

C

```
LIB_EXPORT void laScreen_Delete(laScreen* scr);
```

Returns

void

Parameters

Parameters	Description
laScreen* scr	the screen to destroy

Function

void laScreen_Delete(laScreen* scr)

laScreen_GetHideEventCallback Function

Returns the hide call back event function pointer for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laScreen_ShowHideCallback_FnPtr laScreen_GetHideEventCallback(laScreen* scr);
```

Returns

laScreen_ShowHideCallback_FnPtr

Description

Returns the hide call back event function pointer for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to reference

Function

aScreen_ShowHideCallback_FnPtr laScreen_GetHideEventCallback(laScreen* scr)

laScreen_GetLayerIndex Function

Returns the index of the layer for the screen specified.

File

libaria_screen.h

C

```
LIB_EXPORT int32_t laScreen_GetLayerIndex(laScreen* scr, laLayer* layer);
```

Returns

uint32_t - the index of the layer

Description

Returns the index of the layer for the screen specified.

Parameters

Parameters	Description
laScreen* scr	the screen to reference
laLayer* layer	the layer to search for

Function

int32_t laScreen_GetLayerIndex(laScreen* scr, laLayer* layer)

laScreen_GetOrientation Function

Returns the orientation object associated with the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laScreenOrientation laScreen_GetOrientation(laScreen* scr);
```

Returns

laScreenOrientation - the screen orientation

Description

Returns the orientation object associated with the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to reference

Function

laScreenOrientation laScreen_GetOrientation(laScreen* scr)

laScreen_GetShowEventCallback Function

Returns the show call back event function pointer for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laScreen_ShowHideCallback_FnPtr laScreen_GetShowEventCallback(laScreen* scr);
```

Returns

laScreen_ShowHideCallback_FnPtr

Description

Returns the show call back event function pointer for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to reference

Function

laScreen_ShowHideCallback_FnPtr laScreen_GetShowEventCallback(laScreen* scr)

laScreen_Hide Function

Hide the currently active screen

This function has been deprecated in favor of laContext_SetActiveScreen

File

libaria_screen.h

C

```
LIB_EXPORT GFX_DEPRECATED laResult laScreen_Hide(laScreen* scr);
```

Returns

laResult

Description

The function makes sure that the specified screen is currently active, hides the screen by calling the hide callback function pointer, if the persistent flag is not marked for that screen, delete the screen and free memory. Reset or turn off the Layers allocated for the screen.

Parameters

Parameters	Description
laScreen* scr	the screen to hide

Function

laResult laScreen_Hide(laScreen* scr)

laScreen_New Function

Create a new screen, initialize it to the values specified.

File

libaria_screen.h

C

```
LIB_EXPORT laScreen* laScreen_New(laBool persistent, laBool createAtStartup, laScreen_CreateCallback_FnPtr cb);
```

Returns

void

Description

Create a new screen, initialize it to the values specified. The key properties to specify include screen persistence, call backs for screen creation, initialize the screen to default values either specified through MHGC or manually by user.

Parameters

Parameters	Description
laBool persistent	indicates that the screen should not free the memory of its layers when it is hidden
laBool createAtStartup	indicates that the screen should be created as soon as possible to make its widgets accessible to the application
laScreen_CreateCallback_FnPtr cb	the function that should be called to initialize the screen at a later time

Function

laScreen* laScreen_New(laBool persistent,
laBool createAtStartup,
laScreen_CreateCallback_FnPtr cb)

laScreen_SetHideEventCallback Function

Set the hide call back event function pointer for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetHideEventCallback(laScreen* scr, laScreen_ShowHideCallback_FnPtr cb);
```

Returns

laResult

Description

Set the hide call back event function pointer for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to modify laScreen_ShowHideCallback_FnPtr

Function

laResult laScreen_SetHideEventCallback(laScreen* scr,
laScreen_ShowHideCallback_FnPtr cb)

laScreen_SetLayer Function

Assigns the provided layer pointer to the screen at the given index

This function has been deprecated in favor of laContext_SetActiveScreen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetLayer(laScreen* scr, uint32_t idx, laLayer* layer);
```

Returns

laResult - the result of the operation

Description

Screens contain an internal list of layer pointers. This API assigns a layer to a screen. If the screen is currently active the library attempts to immediately enable the new layer in the HAL.

Parameters

Parameters	Description
laScreen* scr	the screen to modify
uint32_t idx	the index of the layer
laLayer* layer	the layer pointer to assign to the screen

Function

laResult laScreen_SetLayer(laScreen* scr, uint32_t idx, laLayer* layer)

laScreen_SetOrientation Function

Sets the orientation object to the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetOrientation(laScreen* scr, laScreenOrientation ori);
```

Returns

laResult - the result of the operation

Description

Sets the orientation object to the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to modify

laScreenOrientation	the new orientation setting

Function

laResult laScreen_SetOrientation(laScreen* scr, laScreenOrientation ori)

laScreen_SetShowEventCallback Function

Set the show call back event function pointer for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetShowEventCallback(laScreen* scr, laScreen_ShowHideCallback_FnPtr cb);
```

Returns

laResult - the result of the operation

Description

Set the show call back event function pointer for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to modify
laScreen_ShowHideCallback_FnPtr	the function pointer to use

Function

laResult laScreen_SetShowEventCallback(laScreen* scr,
laScreen_ShowHideCallback_FnPtr cb)

laScreen_Show Function

Make the specified screen active and show it on the display

File

libaria_screen.h

C

```
LIB_EXPORT GFX_DEPRECATED laResult laScreen_Show(laScreen* scr);
```

Returns

void

Description

The function makes sure that the specified screen is not already active, creates it if it is is not already created, sets the appropriate color mode, make it active and call the show callback function pointer.

Parameters

Parameters	Description
laScreen* scr	the screen to show

Function

laResult laScreen_Show(laScreen* scr)

laScrollBarWidget_GetExtentValue Function

Gets the current scroll bar extent value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT uint32_t laScrollBarWidget_GetExtentValue(laScrollBarWidget* bar);
```

Returns

uint32_t - the extent value

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

uint32_t laScrollBarWidget_GetExtentValue(laScrollBarWidget* bar)

laScrollBarWidget_GetMaxinumValue Function

Gets the maximum scroll value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT uint32_t laScrollBarWidget_GetMaxinumValue(laScrollBarWidget* bar);
```

Returns

uint32_t - the maximum scroll value

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

uint32_t laScrollBarWidget_GetMaxinumValue(laScrollBarWidget* bar)

laScrollBarWidget_GetOrientation Function

Gets the orientation value for the scroll bar

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laScrollBarOrientation laScrollBarWidget_GetOrientation(laScrollBarWidget* bar);
```

Returns

laScrollBarOrientation - the orientation value

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

laScrollBarOrientation laScrollBarWidget_GetOrientation(laScrollBarWidget* bar)

laScrollBarWidget_GetScrollPercentage Function

Gets the current scroll value as a percentage

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT uint32_t laScrollBarWidget_GetScrollPercentage(laScrollBarWidget* bar);
```

Returns

uint32_t - the scroll percentage

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

uint32_t laScrollBarWidget_GetScrollPercentage(laScrollBarWidget* bar)

laScrollBarWidget_GetScrollValue Function

Gets the current scroll value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT uint32_t laScrollBarWidget_GetScrollValue(laScrollBarWidget* bar);
```

Returns

uint32_t - the scroll value

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

uint32_t laScrollBarWidget_GetScrollValue(laScrollBarWidget* bar)

laScrollBarWidget_GetStepSize Function

Gets the current discreet step size

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT uint32_t laScrollBarWidget_GetStepSize(laScrollBarWidget* bar);
```

Returns

uint32_t - the current step size

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

uint32_t laScrollBarWidget_GetStepSize(laScrollBarWidget* bar)

laScrollBarWidget_GetValueChangedEventCallback Function

Gets the current value changed callback function pointer

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laScrollBarWidget_ValueChangedEvent
laScrollBarWidget_GetValueChangedEventCallback(laScrollBarWidget* bar);
```

Returns

laScrollBarWidget_ValueChangedEvent - a valid pointer or NULL

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

laScrollBarWidget_ValueChangedEvent laScrollBarWidget_GetValueChangedEventCallback(laScrollBarWidget* bar)

laScrollBarWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laScrollBarWidget* laScrollBarWidget_New();
```

Returns

laScrollBarWidget*

Function

laScrollBarWidget* laScrollBarWidget_New()

laScrollBarWidget_SetExtentValue Function

Sets the scroll bar extent value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetExtentValue(laScrollBarWidget* bar, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
uint32_t val	the extent value

Function

laResult laScrollBarWidget_SetExtentValue(laScrollBarWidget* bar,
uint32_t val)

laScrollBarWidget_SetMaximumValue Function

Sets the maximum scroll value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetMaximumValue(laScrollBarWidget* bar, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
uint32_t val	the desired maximum scroll value

Function

laResult laScrollBarWidget_SetMaximumValue(laScrollBarWidget* bar,
uint32_t val)

laScrollBarWidget_SetOrientation Function

Sets the orientation value of the scroll bar

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetOrientation(laScrollBarWidget* bar, laScrollBarOrientation align,
laBool swapDimensions);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
laScrollBarOrientation	the desired orientation value

Function

laResult laScrollBarWidget_SetOrientation(laScrollBarWidget* bar,
laScrollBarOrientation align)

laScrollBarWidget_SetScrollPercentage Function

Sets the current scroll value using a percentage. Percentage should be a value from 0 - 100

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetScrollPercentage(laScrollBarWidget* bar, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
uint32_t val	a value from 0 - 100

Function

laResult laScrollBarWidget_SetScrollPercentage(laScrollBarWidget* bar,
uint32_t val)

laScrollBarWidget_SetScrollValue Function

Sets the current scroll value

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetScrollValue(laScrollBarWidget* bar, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
uint32_t	the desired scroll value

Function

laResult laScrollBarWidget_SetScrollValue(laScrollBarWidget* bar,
uint32_t val)

laScrollBarWidget_SetStepSize Function

Sets the current step size

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetStepSize(laScrollBarWidget* bar, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
uint32_t val	the desired step size

Function

laResult laScrollBarWidget_SetStepSize(laScrollBarWidget* bar,
uint32_t val)

laScrollBarWidget_SetValueChangedEventCallback Function

Sets the value changed event callback pointer

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_SetValueChangedEventCallback(laScrollBarWidget* bar,
laScrollBarWidget_ValueChangedEvent cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget
laScrollBarWidget_ValueChangedEvent	a valid pointer or NULL

Function

laResult laScrollBarWidget_SetValueChangedEventCallback(laScrollBarWidget* bar,
laScrollBarWidget_ValueChangedEvent cb)

laScrollBarWidget_StepBackward Function

Moves the scroll value back by the current step size

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_StepBackward(laScrollBarWidget* bar);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

laResult laScrollBarWidget_StepBackward(laScrollBarWidget* bar)

laScrollBarWidget_StepForward Function

Moves the scroll value forward by the current step size

File

libaria_widget_scrollbar.h

C

```
LIB_EXPORT laResult laScrollBarWidget_StepForward(laScrollBarWidget* bar);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laScrollBarWidget* bar	the widget

Function

laResult laScrollBarWidget_StepForward(laScrollBarWidget* bar)

laSliderWidget_GetGripSize Function

Gets the current grip size of the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT uint32_t laSliderWidget_GetGripSize(laSliderWidget* sld);
```

Returns

uint32_t - the current grip size

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

uint32_t laSliderWidget_GetGripSize(laSliderWidget* sld)

laSliderWidget_GetMaxinumValue Function

Gets the maximum value for the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT uint32_t laSliderWidget_GetMaxinumValue(laSliderWidget* sld);
```

Returns

uint32_t - the maximum value for the slider

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

uint32_t laSliderWidget_GetMaxinumValue(laSliderWidget* sld)

laSliderWidget_GetMininumValue Function

Gets the minimum value for the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT uint32_t laSliderWidget_GetMininumValue(laSliderWidget* sld);
```

Returns

uint32_t - the minimum slider value

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

uint32_t laSliderWidget_GetMininumValue(laSliderWidget* sld)

laSliderWidget_GetOrientation Function

Gets the orientation value for the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT laSliderOrientation laSliderWidget_GetOrientation(laSliderWidget* sld);
```

Returns

laSliderOrientation

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

laSliderOrientation laSliderWidget_GetOrientation(laSliderWidget* sld)

laSliderWidget_GetSliderPercentage Function

Gets the slider value as a percentage

File

libaria_widget_slider.h

C

```
LIB_EXPORT uint32_t laSliderWidget_GetSliderPercentage(laSliderWidget* sld);
```

Returns

uint32_t - the slider value as a percentage

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

uint32_t laSliderWidget_GetSliderPercentage(laSliderWidget* sld)

laSliderWidget_GetSliderValue Function

Gets the current slider value

File

libaria_widget_slider.h

C

```
LIB_EXPORT int32_t laSliderWidget_GetSliderValue(laSliderWidget* sld);
```

Returns

uint32_t - the current slider value

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

uint32_t laSliderWidget_GetSliderValue(laSliderWidget* sld)

laSliderWidget_GetValueChangedEventCallback Function

Gets the current value changed event callback pointer

File

libaria_widget_slider.h

C

```
LIB_EXPORT laSliderWidget_ValueChangedEvent laSliderWidget_GetValueChangedEventCallback(laSliderWidget*
sld);
```

Returns

laSliderWidget_ValueChangedEvent - a valid callback or NULL

Parameters

Parameters	Description
laSliderWidget* sld	the widget

Function

laSliderWidget_ValueChangedEvent laSliderWidget_GetValueChangedEventCallback(laSliderWidget* sld)

laSliderWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_slider.h

C

```
LIB_EXPORT laSliderWidget* laSliderWidget_New();
```

Returns

laSliderWidget*

Function

laSliderWidget* laSliderWidget_New()

laSliderWidget_SetGripSize Function

Sets the grip size of the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetGripSize(laSliderWidget* sld, uint32_t size);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
uint32_t size	the desired grip size

Function

laResult laSliderWidget_SetGripSize(laSliderWidget* sld,

uint32_t size)

laSliderWidget_SetMaximumValue Function

Sets the maximum value for the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetMaximumValue(laSliderWidget* sld, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
uint32_t val	the desired maximum value for the slider

Function

laResult laSliderWidget_SetMaximumValue(laSliderWidget* sld,
uint32_t val)

laSliderWidget_SetMinimumValue Function

Sets the minimum value for the slider

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetMinimumValue(laSliderWidget* sld, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
uint32_t val	the desired minimum value

Function

laResult laSliderWidget_SetMinimumValue(laSliderWidget* sld,
uint32_t val)

laSliderWidget_SetOrientation Function

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetOrientation(laSliderWidget* sld, laSliderOrientation align, laBool
swapDimensions);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget

laSliderOrientation	the desired slider orientation
laBool	indicates if the width and height of the slider should be swapped

Function

laResult laSliderWidget_SetOrientation(laSliderWidget* sld,
laSliderOrientation align,
laBool swapDimensions)

laSliderWidget_SetSliderPercentage Function

Sets the slider value using a percentage. Value must be from 0 - 100.

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetSliderPercentage(laSliderWidget* sld, uint32_t val);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
uint32_t val	a percentage value from 0 - 100

Function

laResult laSliderWidget_SetSliderPercentage(laSliderWidget* sld,
uint32_t val)

laSliderWidget_SetSliderValue Function

Sets the current slider value

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetSliderValue(laSliderWidget* sld, int32_t val);
```

Returns

laResult - the operation result

Description

Must be between slider min and max

Parameters

Parameters	Description
laSliderWidget* sld	the widget
uint32_t val	the desired slider value

Function

laResult laSliderWidget_SetSliderValue(laSliderWidget* sld,
int32_t val)

laSliderWidget_SetValueChangedEventCallback Function

Sets the value changed event callback pointer

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_SetValueChangedEventCallback(laSliderWidget* sld,
laSliderWidget_ValueChangedEvent cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
laSliderWidget_ValueChangedEvent	a valid pointer or NULL

Function

laResult laSliderWidget_SetValueChangedEventCallback(laSliderWidget* sld,
laSliderWidget_ValueChangedEvent cb)

laSliderWidget_Step Function

Moves the slider by a given amount

File

libaria_widget_slider.h

C

```
LIB_EXPORT laResult laSliderWidget_Step(laSliderWidget* sld, int32_t amount);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laSliderWidget* sld	the widget
int32_t amount	the amount by which to adjust the current slider value

Function

laResult laSliderWidget_Step(laSliderWidget* sld, int32_t amount)

laTextFieldWidget_GetAlignment Function

Gets the text horizontal alignment value.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laHAlignment laTextFieldWidget_GetAlignment(laTextFieldWidget* txt);
```

Returns

laHAlignment - the horizontal alignment value

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

Function

laHAlignment laTextFieldWidget_GetAlignment(laTextFieldWidget* txt)

laTextFieldWidget_GetCursorDelay Function

Gets the current cursor delay.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT uint32_t laTextFieldWidget_GetCursorDelay(laTextFieldWidget* txt);
```

Returns

uint32_t - the current delay value

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

Function

uint32_t laTextFieldWidget_GetCursorDelay(laTextFieldWidget* txt)

laTextFieldWidget_GetCursorEnabled Function

Gets the cursor enabled value

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laBool laTextFieldWidget_GetCursorEnabled(laTextFieldWidget* txt);
```

Returns

laBool - the cursor enabled flag value

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

Function

laBool laTextFieldWidget_GetCursorEnabled(laTextFieldWidget* txt)

laTextFieldWidget_GetCursorPosition Function

Gets the current edit cursor position

File

libaria_widget_textfield.h

C

```
LIB_EXPORT uint32_t laTextFieldWidget_GetCursorPosition(laTextFieldWidget* txt);
```

Returns

uint32_t - the index of the cursor

Description

This cursor will appear to the left of the character at index of the string

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

Function

uint32_t laTextFieldWidget_GetCursorPosition(laTextFieldWidget* txt)

laTextFieldWidget_GetText Function

Gets the text value for the box.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_GetText(laTextFieldWidget* txt, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
laString* str	a pointer to an laString object

Function

laResult laTextFieldWidget_GetText(laTextFieldWidget* txt, laString* str)

laTextFieldWidget_GetTextChangedEventCallback Function

Gets the current text changed event callback pointer

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laTextFieldWidget_TextChangedCallback
laTextFieldWidget_GetTextChangedEventCallback(laTextFieldWidget* txt);
```

Returns

laTextFieldWidget_TextChangedCallback - a valid pointer or NULL

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

Function

laTextFieldWidget_TextChangedCallback laTextFieldWidget_GetTextChangedEventCallback(laTextFieldWidget* txt)

laTextFieldWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laTextFieldWidget* laTextFieldWidget_New();
```

Returns

laTextFieldWidget*

Function

laTextFieldWidget* laTextFieldWidget_New()

laTextFieldWidget_SetAlignment Function

Sets the text horizontal alignment value

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetAlignment(laTextFieldWidget* txt, laHAlignment align);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
laHAlignment	the horizontal alignment value

Function

laResult laTextFieldWidget_SetAlignment(laTextFieldWidget* txt,
laHAlignment align)

laTextFieldWidget_SetCursorDelay Function

Sets the cursor delay value

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetCursorDelay(laTextFieldWidget* txt, uint32_t dt);
```

Returns

laResult - the operation result

Description

This value is typically expressed in milliseconds

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
uint32_t dt	the cursor delay value

Function

laResult laTextFieldWidget_SetCursorDelay(laTextFieldWidget* txt,
uint32_t dt)

laTextFieldWidget_SetCursorEnabled Function

Sets the cursor enabled value flag

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetCursorEnabled(laTextFieldWidget* txt, laBool en);
```

Returns

laResult - the operation result

Description

The cursor enabled flag controls whether the cursor will display or not

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
laBool en	the desired flag state

Function

laResult laTextFieldWidget_SetCursorEnabled(laTextFieldWidget* txt,
laBool en)

laTextFieldWidget_SetCursorPosition Function

Sets the position of the cursor

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetCursorPosition(laTextFieldWidget* txt, uint32_t pos);
```

Returns

laResult - the operation result

Description

The cursor will appear to the left of the character at pos

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
uint32_t pos	the position of the cursor in character indices

Function

laResult laTextFieldWidget_SetCursorPosition(laTextFieldWidget* txt,
uint32_t pos)

laTextFieldWidget_SetText Function

Sets the text value for the box.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetText(laTextFieldWidget* txt, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget

laString str	an laString object

Function

laResult laTextFieldWidget_SetText(laTextFieldWidget* txt, laString str)

laTextFieldWidget_SetTextChangedEventCallback Function

Sets the text changed event callback pointer

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetTextChangedEventCallback(laTextFieldWidget* txt,
laTextFieldWidget_TextChangedCallback cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
laTextFieldWidget_TextChangedCallback	a valid pointer or NULL

Function

laResult laTextFieldWidget_SetTextChangedEventCallback(laTextFieldWidget* txt,
laTextFieldWidget_TextChangedCallback cb)

laTouchTest_AddPoint Function

Adds a point to the touch test widget. The point will then be displayed.

File

libaria_widget_touchtest.h

C

```
LIB_EXPORT laResult laTouchTest_AddPoint(laTouchTestWidget* tch, GFX_Point* pnt);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTouchTestWidget* tch	the widget
GFX_Point* pnt	a pointer to the point to add

Function

laResult laTouchTest_AddPoint(laTouchTestWidget* tch, GFX_Point* pnt)

laTouchTest_ClearPoints Function

Clears all of the existing touch points

File

libaria_widget_touchtest.h

C

```
LIB_EXPORT laResult laTouchTest_ClearPoints(laTouchTestWidget* tch);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTouchTestWidget* tch	the widget

Function

laResult laTouchTest_ClearPoints(laTouchTestWidget* tch)

laTouchTestWidget_GetPointAddedEventCallback Function

Gets the current point added event callback

File

libaria_widget_touchtest.h

C

```
LIB_EXPORT laTouchTestWidget_PointAddedEventCallback
laTouchTestWidget_GetPointAddedEventCallback(laTouchTestWidget* txt);
```

Returns

laTouchTestWidget_PointAddedEventCallback - a valid pointer or NULL

Parameters

Parameters	Description
laTouchTestWidget* tch	the widget

Function

laTouchTestWidget_PointAddedEventCallback laTouchTestWidget_GetPointAddedEventCallback(laTouchTestWidget* txt)

laTouchTestWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_touchtest.h

C

```
LIB_EXPORT laTouchTestWidget* laTouchTestWidget_New();
```

Returns

laTouchTestWidget*

Function

laTouchTestWidget* laTouchTestWidget_New()

laTouchTestWidget_SetPointAddedEventCallback Function

Sets the point added event callback

File

libaria_widget_touchtest.h

C

```
LIB_EXPORT laResult laTouchTestWidget_SetPointAddedEventCallback(laTouchTestWidget* txt,
laTouchTestWidget_PointAddedEventCallback cb);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTouchTestWidget* tch	the widget
laTouchTestWidget_PointAddedEventCallback cb	a valid pointer or NULL

Function

laResult laTouchTestWidget_SetPointAddedEventCallback(laTouchTestWidget* txt,
laTouchTestWidget_PointAddedEventCallback cb)

laUtils_ArrangeRectangle Function

Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the image rectangle.

File

libaria_utils.h

C

```
void laUtils_ArrangeRectangle(GFX_Rect* sub, GFX_Rect obj, GFX_Rect bounds, laHAlignment hAlignment,
laVAlignment vAlignment, laRelativePosition position, uint8_t leftMargin, uint8_t topMargin, uint8_t
rightMargin, uint8_t bottomMargin, uint16_t rectMargin);
```

Returns

void

Remarks

The x and y position of sub will be manipulated by this function. The dimensions of the rectangle should be set before calling and should remain unchanged after execution.

Parameters

Parameters	Description
GFX_Rect* sub	the bounds of the subject rectangle (image)
GFX_Rect obj	the bounds of the object rectangle (text)
GFX_Rect bounds	the bounds of the bounding rectangle (widget)
laHAlignment hAlignment	the horizontal alignment of the rects
laVAlignment vAlignment	the vertical alignment of the rects
laRelativePosition position	the relative position of the rectangles
uint8_t leftMargin	the left margin of the bounding rectangle
uint8_t topMargin	the top margin of the bounding rectangle
uint8_t rightMargin	the right margin of the bounding rectangle
uint8_t bottomMargin	the bottom margin of the bounding rectangle
uint16_t rectMargin	the distance between the image and the text rects

Function

void laUtils_ArrangeRectangle(GFX_Rect* sub,
GFX_Rect obj,
GFX_Rect bounds,
laHAlignment hAlignment,
laVAlignment vAlignment,
laRelativePosition position,
uint8_t leftMargin,
uint8_t topMargin,
uint8_t rightMargin,
uint8_t bottomMargin,
uint16_t rectMargin)

laUtils_ArrangeRectangleRelative Function

Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the text rectangle.

File

libaria_utils.h

C

```
void laUtils_ArrangeRectangleRelative(GFX_Rect* sub, GFX_Rect obj, GFX_Rect bounds, laHAlignment
hAlignment, laVAlignment vAlignment, laRelativePosition position, uint8_t leftMargin, uint8_t topMargin,
uint8_t rightMargin, uint8_t bottomMargin, uint16_t rectMargin);
```

Returns

void

Remarks

The x and y position of sub will be manipulated by this function. The dimensions of the rectangle should be set before calling and should remain unchanged after execution.

Parameters

Parameters	Description
GFX_Rect* sub	the bounds of the subject rectangle (text)
GFX_Rect obj	the bounds of the object rectangle (image)
GFX_Rect bounds	the bounds of the bounding rectangle (widget)
laHAlignment hAlignment	the horizontal alignment of the rects
laVAlignment vAlignment	the vertical alignment of the rects
laRelativePosition position	the relative position of the rectangles
uint8_t leftMargin	the left margin of the bounding rectangle
uint8_t topMargin	the top margin of the bounding rectangle
uint8_t rightMargin	the right margin of the bounding rectangle
uint8_t bottomMargin	the bottom margin of the bounding rectangle
uint16_t rectMargin	the distance between the image and the text rects

Function

void laUtils_ArrangeRectangleRelative(GFX_Rect* sub,
GFX_Rect obj,
GFX_Rect bounds,
laHAlignment hAlignment,
laVAlignment vAlignment,
laRelativePosition position,
uint8_t leftMargin,
uint8_t topMargin,
uint8_t rightMargin,
uint8_t bottomMargin,
uint16_t rectMargin)

laUtils_ChildIntersectsParent Function

Performs an intersection test between a parent widget and a child widget

File

libaria_utils.h

C

```
laBool laUtils_ChildIntersectsParent(laWidget* parent, laWidget* child);
```

Returns

laBool - result of the intersection test

Parameters

Parameters	Description
laWidget* parent	the parent widget
laWidget* child	the child widget

Function

laBool laUtils_ChildIntersectsParent(laWidget* parent, laWidget* child)

laUtils_ClipRectToParent Function

Clips a rectangle to the parent of a widget

File

libaria_utils.h

C

```
void laUtils_ClipRectToParent(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to clip

Function

void laUtils_ClipRectToParent(laWidget* widget, GFX_Rect* rect)

laUtils_GetLayer Function

Finds the root parent of a widget, which should be a layer

File

libaria_utils.h

C

```
laLayer* laUtils_GetLayer(laWidget* widget);
```

Returns

laLayer* - the widget's owning layer

Parameters

Parameters	Description
laWidget* widget	the subject widget

Function

laLayer* laUtils_GetLayer(laWidget* widget)

laUtils_ListOcclusionCullTest Function

Performs an occlusion test on a list of widgets an a rectangular area. This attempts to find only the topmost widgets for the given area. If a widget is completely occluded then it is removed from the list. Any widgets that remain in the list should be redrawn by the rasterizer.

File

libaria_utils.h

C

```
void laUtils_ListOcclusionCullTest(laList* list, GFX_Rect rect);
```

Returns

void

Parameters

Parameters	Description
laList* list	the widget list to test
GFX_Rect rect	the occlusion area

Function

void laUtils_ListOcclusionCullTest(laList* list, GFX_Rect rect)

laUtils_OcclusionCullTest Function

Performs an occlusion test for a widget in the tree. A widget is occluded if it is completely covered by one or more widgets. This is useful for culling widgets before the rasterizing phase.

File

libaria_utils.h

C

```
laBool laUtils_OcclusionCullTest(laWidget* widget);
```

Returns

laBool - result of the occlusion test

Parameters

Parameters	Description
laWidget* widget	the widget to test

Function

laBool laUtils_OcclusionCullTest(laWidget* widget)

laUtils_Pick Function

Finds the top-most visible widget in the tree at the given coordinates.

File

libaria_utils.h

C

```
LIB_EXPORT laWidget* laUtils_Pick(int32_t x, int32_t y);
```

Returns

laWidget* - the result widget

Parameters

Parameters	Description
int32_t x	the x coordinate of the pick point
int32_t y	the y coordinate of the pick point

Function

laWidget* laUtils_Pick(int32_t x, int32_t y)

laUtils_PickRect Function

Finds all of the visible widgets in the given rectangular area.

File

libaria_utils.h

C

```
void laUtils_PickRect(laLayer* layer, GFX_Rect rect, laList* list);
```

Returns

void

Parameters

Parameters	Description
laLayer* layer	the layer to analyze
GFX_Rect	the rectangle pick area
laList* list	the result list

Function

void laUtils_PickRect(laLayer* layer, GFX_Rect rect, laList* list)

laUtils_PointScreenToLocalSpace Function

Converts a point from layer space into the local space of a widget

File

libaria_utils.h

C

```
void laUtils_PointScreenToLocalSpace(laWidget* widget, GFX_Point* pnt);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Point* pnt	the point to convert

Function

void laUtils_PointLayerToLocalSpace(laWidget* widget, GFX_Point* pnt)

laUtils_RectFromParentSpace Function

Converts a rectangle from widget parent space to widget local space

File

libaria_utils.h

C

```
void laUtils_RectFromParentSpace(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to convert

Function

void laUtils_RectFromParentSpace(laWidget* widget, GFX_Rect* rect)

laUtils_RectToLayerSpace Function

Converts a rectangle from widget local space to layer space

File

libaria_utils.h

C

```
void laUtils_RectToLayerSpace(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to convert

Function

void laUtils_RectToLayerSpace(laWidget* widget, GFX_Rect* rect)

laUtils_RectToParentSpace Function

Converts a rectangle from widget local space to widget parent space. Widget must be a child of a layer for this to function.

File

libaria_utils.h

C

```
void laUtils_RectToParentSpace(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to convert

Function

void laUtils_RectToParentSpace(laWidget* widget, GFX_Rect* rect)

laUtils_RectToScreenSpace Function

Converts a rectangle from widget local space to screen space

File

libaria_utils.h

C

```
void laUtils_RectToScreenSpace(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to convert

Function

void laUtils_RectToScreenSpace(laWidget* widget, GFX_Rect* rect)

laWidget_AddChild Function

Adds the child to the parent widget specified in the argument

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_AddChild(laWidget* parent, laWidget* child);
```

Returns

laResult - the operation result

Description

The function checks to see if the child and parent are valid, removes the child from its current parents children list, and assigns the child to the parent widget specified. The child is attached at the end of the list of the parent widgets children list.

Parameters

Parameters	Description
laWidget* parent	the parent widget
laWidget* child	the child to add

Function

laResult laWidget_AddChild(laWidget* parent, laWidget* child)

laWidget_Delete Function

Delete the widget object specified

File

libaria_widget.h

C

```
LIB_EXPORT void laWidget_Delete(laWidget* wgt);
```

Returns

void

Description

Delete a widget object specified, de-allocate memory for the widget through the current active context. All child widgets are also destructed and freed.

Function

void laWidget_Delete(laWidget* wgt)

laWidget_GetAlphaAmount Function

Return the widget's global alpha amount

File

libaria_widget.h

C

```
LIB_EXPORT uint32_t laWidget_GetAlphaAmount(laWidget* wgt);
```

Returns

uint32_t - the widget's global alpha amount

Description

Return the widget's global alpha amount

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

uint32_t laWidget_GetAlphaAmount(laWidget* wgt)

laWidget_GetAlphaEnable Function

Return the alpha enable property of the widget

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_GetAlphaEnable(laWidget* wgt);
```

Returns

laBool - the widget's alpha enable flag value

Description

Return the alpha enable property of the widget

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

laBool laWidget_GetAlphaEnable(laWidget* wgt)

laWidget_GetBorderType Function

Return the border type associated with the widget object

File

libaria_widget.h

C

```
LIB_EXPORT laBorderType laWidget_GetBorderType(laWidget* wgt);
```

Returns

laBorderType - the current widget border type

Description

Return the border type associated with the widget object

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBorderType laWidget_GetBorderType(laWidget* wgt)

laWidget_GetChildAtIndex Function

Fetches the child at the specified index from the children list of the specified parent widget

File

libaria_widget.h

C

```
LIB_EXPORT laWidget* laWidget_GetChildAtIndex(laWidget* parent, uint32_t idx);
```

Returns

laWidget* - a valid child pointer or NULL

Description

Fetches the child at the specified index from the children list of the specified parent widget

Parameters

Parameters	Description
laWidget* wgt	the widget
uint32_t idx	the desired child index

Function

laWidget* laWidget_GetChildAtIndex(laWidget* parent, uint32_t idx)

laWidget_GetChildCount Function

Returns the size of the children list of the specified parent widget

File

libaria_widget.h

C

```
LIB_EXPORT uint32_t laWidget_GetChildCount(laWidget* parent);
```

Returns

uint32_t - the number of children of this widget

Description

Returns the size of the children list of the specified parent widget

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

uint32_t laWidget_GetChildCount(laWidget* parent)

laWidget_GetCumulativeAlphaAmount Function

Calculates the cumulative alpha amount for a hierarchy of widgets

File

libaria_widget.h

C

```
LIB_EXPORT uint32_t laWidget_GetCumulativeAlphaAmount(laWidget* wgt);
```

Returns

uint32_t - the cumulative blending amount

Description

Alpha blending amounts are cumulative from parent to child. If a parent is blended at 50% then logically a child should also implicitely be blended at 50%. If a child further explictely enables blending at 50% then the cumulative amount is 25%.

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

uint32_t laWidget_GetCumulativeAlphaAmount(laWidget* wgt)

laWidget_GetCumulativeAlphaEnable Function

Determines if this or any ancestor widget has alpha enabled

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_GetCumulativeAlphaEnable(laWidget* wgt);
```

Returns

laBool - whether the widget has alpha enabled

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

laBool laWidget_GetCumulativeAlphaEnable(laWidget* wgt)

laWidget_GetEnabled Function

Returns the boolean value of the widget enabled property

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_GetEnabled(laWidget* wgt);
```

Returns

laBool - the value of the enabled flag

Description

Returns the boolean value of the widget enabled property. The widget enable flag often governs things like appearing 'greyed out' and prohibits user interactiong if it is false. Widgets must individually support this flag.

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBool laWidget_GetEnabled(laWidget* wgt)

laWidget_GetHeight Function

Returns the widget rectangles height

File

libaria_widget.h

C

```
LIB_EXPORT int32_t laWidget_GetHeight(laWidget* wgt);
```

Returns

uint32_t - the widget's width value

Description

Returns the widget rectangles height

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

int32_t laWidget_GetHeight(laWidget* wgt)

laWidget_GetIndexOfChild Function

Fetches the index of the child from the children list of the specified parent widget

File

libaria_widget.h

C

```
LIB_EXPORT int32_t laWidget_GetIndexOfChild(laWidget* parent, laWidget* child);
```

Returns

int32_t - the index of the given child pointer or -1 if not found

Description

Traverses the children list of the specified parent widget and finds the index of the child widget specified.

Parameters

Parameters	Description
laWidget* parent	the parent widget
laWidget* child	the child widget

Function

int32_t laWidget_GetIndexOfChild(laWidget* parent, laWidget* child)

laWidget_GetMargin Function

Returns the margin value associated with the widget in the laMargin pointer

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_GetMargin(laWidget* wgt, laMargin* mg);
```

Returns

laResult - the operation result

Description

Returns the margin value associated with the widget in the laMargin pointer

Parameters

Parameters	Description
laWidget* wgt	the widget
laMargin* mg	a pointer to an laMargin object to store the margin values

Function

laResult laWidget_GetMargin (laWidget* wgt, laMargin* mg)

laWidget_GetScheme Function

Returns the scheme associated with the specified widget

File

libaria_widget.h

C

```
LIB_EXPORT laScheme* laWidget_GetScheme(laWidget* wgt);
```

Returns

laScheme* - a pointer to the active scheme for a widget

Description

Returns the scheme associated with the specified widget

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laScheme* laWidget_GetScheme(laWidget* wgt)

laWidget_GetVisible Function

Returns the boolean value of the widget visible property

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_GetVisible(laWidget* wgt);
```

Returns

laBool - the flag value

Description

Returns the boolean value of the widget visible property. Widgets that are invisible will be skipped during the rendering phase. All descendants also logically become invisible when an ancestor does.

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBool laWidget_GetVisible(laWidget* wgt)

laWidget_GetWidth Function

Returns the widget rectangles width

File

libaria_widget.h

C

```
LIB_EXPORT int32_t laWidget_GetWidth(laWidget* wgt);
```

Returns

uint32_t - the widget's y coordinate value

Description

Returns the widget rectangles width

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

int32_t laWidget_GetWidth(laWidget* wgt)

laWidget_GetX Function

Returns the widget rectangles upper left corner x-coordinate

File

libaria_widget.h

C

```
LIB_EXPORT int32_t laWidget_GetX(laWidget* wgt);
```

Returns

uint32_t

Description

Returns the widget rectangles upper left corner x-coordinate

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

int32_t laWidget_GetX(laWidget* wgt)

laWidget_GetY Function

Returns the widget rectangles upper left corner y-coordinate

File

libaria_widget.h

C

```
LIB_EXPORT int32_t laWidget_GetY(laWidget* wgt);
```

Returns

uint32_t - the y value

Description

Returns the widget rectangles upper left corner y-coordinate

Parameters

Parameters	Description
lawidget* wgt	the widget

Function

int32_t laWidget_GetY(laWidget* wgt)

laWidget_HasFocus Function

Checks if the widget specified has focus in the current context

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_HasFocus(laWidget* wgt);
```

Returns

laBool - true of the widget currently has context focus

Description

Checks if the widget specified has focus in the current context

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBool laWidget_HasFocus(laWidget* wgt)

laWidget_Invalidate Function

Invalidates the specified widget.

File

libaria_widget.h

C

```
LIB_EXPORT void laWidget_Invalidate(laWidget* wgt);
```

Returns

void

Description

This function invalidates the specified widget. Invalid widgets are redrawn during the next paint loop call. This function may also invalidate the widget's parent, siblings, ancestors, or cousins.

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

void laWidget_Invalidate(laWidget* wgt)

laWidget_isOpaque Function

Returns true if the widget is considered opaque.

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_isOpaque(laWidget* wgt);
```

Returns

laBool - true if the widget is fully opaque

Description

Opacity is determined by a number of factors including: cumulative alpha amount, background type, and the opaque optimization flag.

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBool laWidget_isOpaque(laWidget* wgt)

laWidget_New Function

Create a new widget.

File

libaria_widget.h

C

```
LIB_EXPORT laWidget* laWidget_New();
```

Returns

lawidget*

Description

Create a new widget, alocate memory for the widget through the current active context. Returns a widget object pointer. Application is responsible for managing the widget pointer until the widget is added to a widget tree.

Function

laWidget* laWidget_New()

laWidget_OverrideTouchDownEvent Function

Replace the TouchDownEvent callback for the widget with the new function pointer specified

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_OverrideTouchDownEvent(laWidget* wgt, laWidget_TouchDownEvent_FnPtr ptr);
```

Returns

laResult - the operation result

Description

This function will replace the current touch down event handler for a widget. Widgets may have their own internal override for this function and replacing it will break their internal capabilties.

Parameters

Parameters	Description
laWidget* wgt	the widget
laWidget_TouchDownEvent_FnPtr	a valid pointer or NULL

Function

laResult laWidget_OverrideTouchDownEvent(laWidget* wgt,
laWidget_TouchDownEvent_FnPtr ptr)

laWidget_OverrideTouchMovedEvent Function

Replace the TouchMovedEvent callback for the widget with the new function pointer specified

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_OverrideTouchMovedEvent(laWidget* wgt, laWidget_TouchMovedEvent_FnPtr ptr);
```

Returns

laResult - the operation result

Description

This function will replace the current touch moved event handler for a widget. Widgets may have their own internal override for this function and replacing it will break their internal capabilties.

Parameters

Parameters	Description
laWidget* wgt	the widget
laWidget_TouchMovedEvent_FnPtr	a valid pointer or NULL

Function

laResult laWidget_OverrideTouchMovedEvent(laWidget* wgt,
laWidget_TouchMovedEvent_FnPtr ptr)

laWidget_OverrideTouchUpEvent Function

Replace the TouchUpEvent callback for the widget with the new function pointer specified

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_OverrideTouchUpEvent(laWidget* wgt, laWidget_TouchUpEvent_FnPtr ptr);
```

Returns

laResult - the operation result

Description

This function will replace the current touch up event handler for a widget. Widgets may have their own internal override for this function and replacing it will break their internal capabilties.

Parameters

Parameters	Description
laWidget* wgt	the widget
laWidget_TouchUpEvent_FnPtr	a valid pointer or NULL

Function

laResult laWidget_OverrideTouchUpEvent(laWidget* wgt,
laWidget_TouchUpEvent_FnPtr ptr)

laWidget_RectToLayerSpace Function

Transforms a widget rectangle from local space to its root layer space.

File

libaria_widget.h

C

```
LIB_EXPORT GFX_Rect laWidget_RectToLayerSpace(laWidget* wgt);
```

Returns

GFX_Rect - the transformed rectangle

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

GFX_Rect laWidget_RectToLayerSpace(laWidget* wgt)

laWidget_RectToParentSpace Function

Returns the rectangle containing the parent of the widget specified

File

libaria_widget.h

C

```
LIB_EXPORT GFX_Rect laWidget_RectToParentSpace(laWidget* wgt);
```

Returns

GFX_Rect - the widget rectangle in parent space

Description

Returns the rectangle containing the parent of the widget specified If the widget and the parent are not null, the rectangle defining the parent widget with its upper left corner x and y coordinates is returned

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

GFX_Rect laWidget_RectToParentSpace(laWidget* wgt)

laWidget_RectToScreenSpace Function

Transforms a widget rectangle from local space to screen space coordinates.

File

libaria_widget.h

C

```
LIB_EXPORT GFX_Rect laWidget_RectToScreenSpace(laWidget* wgt);
```

Returns

GFX_Rect - the transformed rectangle

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

GFX_Rect laWidget_RectToScreenSpace(laWidget* wgt)

laWidget_RemoveChild Function

Removes the child from the parent widget specified in the argument

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_RemoveChild(laWidget* parent, laWidget* child);
```

Returns

laResult - the operation result

Description

The function checks to see if the child and parent are valid, removes the child from its current parents children list

Parameters

Parameters	Description
laWidget* parent	the parent widget
laWidget* child	the child to remove

Function

laResult laWidget_RemoveChild(laWidget* parent, laWidget* child)

laWidget_Resize Function

Changes the widget size by the new defined width and height increments.

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_Resize(laWidget* wgt, int32_t width, int32_t height);
```

Returns

laResult - the operation result

Description

Changes the widget size by the new defined width and height increments.

Parameters

Parameters	Description
laWidget* wgt	the widget
int32_t width	the amount to change the width by
int32_t height	the amount ot change the height by

Function

void laWidget_Resize(laWidget* wgt, int32_t width, int32_t height)

laWidget_SetAlphaAmount Function

Set the widget's global alpha amount to the specified alpha amount

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetAlphaAmount(laWidget* wgt, uint32_t alpha);
```

Returns

laResult - the result of the operation

Description

Set the widget's global alpha amount to the specified alpha amount. Widgets may enable alpha blending even for color modes that don't support an alpha channel.

Parameters

Parameters	Description
laWidget* wgt	the widget
uint32_t alpha	the desired global alpha amount

Function

laResult laWidget_SetAlphaAmount(laWidget* wgt, uint32_t alpha)

laWidget_SetAlphaEnable Function

Set the alpha enable property of the widget with the boolean value specified

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetAlphaEnable(laWidget* wgt, laBool enable);
```

Returns

laResult - the result of the operation

Description

Set the alpha enable property of the widget with the boolean value specified

Parameters

Parameters	Description
lawidget* wgt	the widget
laBool enable	the desired alpha enable flag value

Function

laResult laWidget_SetAlphaEnable(laWidget* wgt, laBool enable)

laWidget_SetBorderType Function

Set the border type associated with the widget object

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetBorderType(laWidget* wgt, laBorderType type);
```

Returns

laResult - the operation result

Description

Set the border type associated with the widget object

Parameters

Parameters	Description
laWidget* wgt	the widget
laBorderType type	the desired border type

Function

laResult laWidget_SetBorderType(laWidget* wgt, laBorderType type)

laWidget_SetEnabled Function

Sets the boolean value of the widget enabled property

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetEnabled(laWidget* wgt, laBool enable);
```

Returns

laResult - the operation result

Description

Sets the boolean value of the widget enabled property

Parameters

Parameters	Description
laWidget* wgt	the widget
laBool	the desired enabled flag value

Function

laResult laWidget_SetEnabled(laWidget* wgt, laBool enable)

laWidget_SetFocus Function

Set the widget into focus for the current context.

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetFocus(laWidget* wgt);
```

Returns

laResult - the operation result

Description

Set the widget into focus for the current context. The input events etc are received by the widget once it is in focus

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laResult laWidget_SetFocus(laWidget* wgt)

laWidget_SetHeight Function

Sets the widget's height value

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetHeight(laWidget* wgt, int32_t height);
```

Returns

laResult - result of the operation

Description

Sets the widget's height value

Parameters

Parameters	Description
lawidget* wgt	the widget
int32_t height	the desired height value, must be > 0

Function

laResult laWidget_SetHeight(laWidget* wgt, int32_t height)

laWidget_SetMargins Function

Set the margin value for left, right, top and bottom margins associated with the widget

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetMargins(laWidget* wgt, uint32_t left, uint32_t top, uint32_t right,
uint32_t bottom);
```

Returns

laResult - the operation result

Description

Set the margin value for left, right, top and bottom margins associated with the widget. Margins are a generic property and it is up to the individual widget to implement them (or not).

Parameters

Parameters	Description
laWidget* wgt	the widget
uint32_t left	the left margin value
uint32_t top	the top margin value
uint32_t right	the right margin value
uint32_t bottom	the bottom margin value

Function

laResult laWidget_SetMargins(laWidget* wgt,
uint32_t left,
uint32_t top,
uint32_t right,
uint32_t bottom)

laWidget_SetParent Function

Sets the parent of the child widget to that specified in the argument list

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetParent(laWidget* wgt, laWidget* parent);
```

Returns

laResult - the operation result

Description

The function checks to see if the child and parent are valid, removes the child from its current parents children list, and assigns the child to the parent widget specified. The child is attached at the end of the list of the parent widgets children list.

Parameters

Parameters	Description
laWidget* wgt	the widget
laWidget* parent	the desired parent widget

Function

laResult laWidget_SetParent(laWidget* wgt, laWidget* parent)

laWidget_SetPosition Function

Changes the widget position to the new defined x and y coordinates.

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetPosition(laWidget* wgt, int32_t x, int32_t y);
```

Returns

laResult - the operation result

Description

Changes the widget position to the new defined x and y coordinates. Moving widgets can be expensive as it needs to repaint multiple areas of its parent widget.

Parameters

Parameters	Description
laWidget* wgt	the widget
int32_t x	the new x coordinate
int32_t y	the new y coordinate

Function

void laWidget_SetPosition(laWidget* wgt, int32_t x, int32_t y)

laWidget_SetScheme Function

Sets the scheme variable for the specified widget

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetScheme(laWidget* wgt, laScheme* scheme);
```

Returns

laResult - the operation result

Description

Sets the scheme variable for the specified widget. The scheme defines the appearance of the widget. Setting this to NULL may result in undefined behavior if the widget doesn't properly support a NULL scheme.

Parameters

Parameters	Description
laWidget* wgt	the widget
laScheme* scheme	a pointer to a scheme or NULL

Function

void laWidget_SetScheme(laWidget* wgt, laScheme* scheme)

laWidget_SetSize Function

Changes the widget size to the new defined width and height dimensions.

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetSize(laWidget* wgt, uint32_t width, uint32_t height);
```

Returns

laResult - the operation result

Description

Changes the widget size to the new width and height dimensions.

Parameters

Parameters	Description
laWidget* wgt	the widget
int32_t width	the new width size
int32_t height	the new height size

Function

void laWidget_SetSize(laWidget* wgt, uint32_t width, uint32_t height)

laWidget_SetVisible Function

Sets the boolean value of the widget visible property

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetVisible(laWidget* wgt, laBool visible);
```

Returns

laResult - the operation result

Description

Sets the boolean value of the widget visible property

Remarks

This value has no effect on layer objects. Use laLayer_SetEnabled instead.

Parameters

Parameters	Description
laWidget* wgt	the widget
laBool	the desired setting

Function

laResult laWidget_SetVisible(laWidget* wgt, laBool visible)

laWidget_SetWidth Function

Sets the widget's width value

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetWidth(laWidget* wgt, int32_t width);
```

Returns

laResult - result of the operation

Description

Sets the widget's width value

Parameters

Parameters	Description
lawidget* wgt	the widget
int32_t width	the desired width value, must be > 0

Function

laResult laWidget_SetWidth(laWidget* wgt, int32_t width)

laWidget_SetX Function

Sets the widget's x coordinate position

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetX(laWidget* wgt, int32_t x);
```

Returns

laResult - result of the operation

Description

Sets the widget's x coordinate position

Parameters

Parameters	Description
lawidget* wgt	the widget
int32_t x	the desired x value

Function

laResult laWidget_SetX(laWidget* wgt, int32_t x)

laWidget_SetY Function

Sets the widget's y coordinate position

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetY(laWidget* wgt, int32_t y);
```

Returns

laResult - result of the operation

Description

Sets the widget's y coordinate position

Parameters

Parameters	Description
lawidget* wgt	the widget
int32_t y	the desired y value

Function

laResult laWidget_SetY(laWidget* wgt, int32_t y)

laWidget_Translate Function

Changes the widget position by moving the widget by the defined x and y increments.

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_Translate(laWidget* wgt, int32_t x, int32_t y);
```

Returns

laResult - the operation result

Description

Changes the widget position by moving the widget by the defined x and y increments.Moving widgets can be expensive as it needs to repaint multiple areas of its parent widget.

Parameters

Parameters	Description
laWidget* wgt	the widget
int32_t x	the amount to move in x
int32_t y	the amount to move in y

Function

void laWidget_Translate(laWidget* wgt, int32_t x, int32_t y)

laWindowWidget_GetIcon Function

Gets the currently used window icon

File

libaria_widget_window.h

C

```
LIB_EXPORT GFXU_ImageAsset* laWindowWidget_GetIcon(laWindowWidget* win);
```

Returns

GFXU_ImageAsset*

Parameters

Parameters	Description
laWindowWidget* win	the widget

Function

GFXU_ImageAsset* laWindowWidget_GetIcon(laWindowWidget* win)

laWindowWidget_GetIconMargin Function

Gets the current image icon margin

File

libaria_widget_window.h

C

```
LIB_EXPORT uint32_t laWindowWidget_GetIconMargin(laWindowWidget* win);
```

Returns

uint32_t - the icon margin

Parameters

Parameters	Description
laWindowWidget* win	the widget

Function

uint32_t laWindowWidget_GetIconMargin(laWindowWidget* win)

laWindowWidget_GetTitle Function

Gets the title text for this window.

File

libaria_widget_window.h

C

```
LIB_EXPORT laResult laWindowWidget_GetTitle(laWindowWidget* win, laString* str);
```

Returns

laResult - the operation result

Description

This function allocates memory and initializes the input string pointer. The caller is responsible for managing the memory once this function returns.

Parameters

Parameters	Description
laWindowWidget* win	the widget
laString* str	a pointer to an laString object

Function

laResult laWindowWidget_GetTitle(laWindowWidget* win, laString* str)

laWindowWidget_New Function

Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

File

libaria_widget_window.h

C

```
LIB_EXPORT laWindowWidget* laWindowWidget_New();
```

Returns

laWindowWidget*

Function

laWindowWidget* laWindowWidget_New()

laWindowWidget_SetIcon Function

Sets the image to be used as a window icon

File

libaria_widget_window.h

C

```
LIB_EXPORT laResult laWindowWidget_SetIcon(laWindowWidget* win, GFXU_ImageAsset* img);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laWindowWidget* win	the widget
GFXU_ImageAsset*	pointer to an image asset

Function

laResult laWindowWidget_SetIcon(laWindowWidget* win,
GFXU_ImageAsset* img)

laWindowWidget_SetIconMargin Function

Sets the image icon margin

File

libaria_widget_window.h

C

```
LIB_EXPORT laResult laWindowWidget_SetIconMargin(laWindowWidget* win, uint32_t mg);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laWindowWidget* win	the widget
uint32_t mg	the image icon margin value

Function

laResult laWindowWidget_SetIconMargin(laWindowWidget* win, uint32_t mg)

laWindowWidget_SetTitle Function

Sets the title text for the window.

File

libaria_widget_window.h

C

```
LIB_EXPORT laResult laWindowWidget_SetTitle(laWindowWidget* win, laString str);
```

Returns

laResult - the operation result

Description

This function copies the contents of the input string into its internal string buffer. The input string can then be freed or altered without affecting the label's internal string value.

Parameters

Parameters	Description
laWindowWidget* win	the widget
laString str	an laString object

Function

laResult laWindowWidget_SetTitle(laWindowWidget* win, laString str)

laCheckBoxWidget_SetImageMargin Function

Sets the distance between the image and the text

File

libaria_widget_checkbox.h

C

```
LIB_EXPORT laResult laCheckBoxWidget_SetImageMargin(laCheckBoxWidget* btn, uint32_t mg);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laCheckBoxWidget* cbox	the widget
uint32_t mg	the desired image margin value

Function

laResult laCheckBoxWidget_SetImageMargin(laCheckBoxWidget* btn, uint32_t mg)

laContext_SetPreemptionLevel Function

Set the preemption level to the specified value

File

libaria_context.h

C

```
LIB_EXPORT laResult laContext_SetPreemptionLevel(laPreemptionLevel level);
```

Returns

laResult

Description

Set the preemption level to the specified value

Function

laResult laContext_SetPreemptionLevel(laPreemptionLevel level)

laKeyPadWidget_SetKeyBackgroundType Function

Sets the background type for a key pad cell at row/column

File

libaria_widget_keypad.h

C

```
LIB_EXPORT laResult laKeyPadWidget_SetKeyBackgroundType(laKeyPadWidget* pad, uint32_t row, uint32_t col,
laBackgroundType type);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laKeyPadWidget* pad	the widget
uint32_t row	the indicated row
uint32_t col	the indicated column
laBackgroundType type	the desired background type

Function

laResult laKeyPadWidget_SetKeyBackgroundType(laKeyPadWidget* pad,
uint32_t row,
uint32_t col,
laBackgroundType type)

laLayer_GetAllowInputPassThrough Function

Gets the layer's input passthrough setting

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetAllowInputPassThrough(const laLayer* layer);
```

Returns

laBool - the state of the layer's passthrough flag

Description

The input passthrough setting is used to prohibit or allow input events to pass through a layer. If a layer is opaque or semi-opaque input events should probably not be allowed to pass through. If the layer is completely transparent then input events may be allowed to pass through to interact with widgets on layers further back in the hierarchy.

An application that disables this is responsible for ensuring that it is modified when the dimensions of the layer change.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

laBool laLayer_GetAllowInputPassThrough(const laLayer* layer)

laLayer_GetEnabled Function

Returns the boolean value of the layer enabled property

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetEnabled(const laLayer* layer);
```

Returns

laBool - the flag value

Description

Returns the boolean value of the layer enabled property

Parameters

Parameters	Description
laLayer*	the layer

Function

laBool laLayer_GetEnabled(const laLayer* layer)

laLayer_SetAllowInputPassthrough Function

Sets the layer's input passthrough flag.

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetAllowInputPassthrough(laLayer* layer, laBool enable);
```

Returns

laResult - the result of the operation

Description

Sets the layer's input passthrough flag.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

laResult laLayer_SetAllowInputPassthrough(laLayer* layer, laBool enable)

laLayer_SetEnabled Function

Sets the boolean value of the layer enabled property

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetEnabled(laLayer* widget, laBool enable);
```

Returns

laResult - the result of the operation

Description

Sets the boolean value of the layer enabled property

Remarks

The enabled flag for a layer will often control the hardware setting for layer usage, depending on the display driver

Parameters

Parameters	Description
laLayer*	the layer
laBool	the desired enabled value

Function

laResult laLayer_SetEnabled(laLayer* widget, laBool enable)

laListWheelWidget_GetFlickInitSpeed Function

Returns the flick init speed for the wheel.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetFlickInitSpeed(laListWheelWidget* whl);
```

Returns

uint32_t - the flick init speed value

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetFlickInitSpeed(laListWheelWidget* whl)

laListWheelWidget_GetIndicatorArea Function

Returns the spacing for the selected item indicator bars.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetIndicatorArea(laListWheelWidget* whl);
```

Returns

uint32_t - the display area

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetIndicatorArea(laListWheelWidget* whl)

laListWheelWidget_GetMaxMomentum Function

Returns the maximum momentum value for the wheel.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetMaxMomentum(laListWheelWidget* whl);
```

Returns

uint32_t - the maximum momentum value.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetMaxMomentum(laListWheelWidget* whl)

laListWheelWidget_GetMomentumFalloffRate Function

Returns the momentum falloff rate for the wheel.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetMomentumFalloffRate(laListWheelWidget* whl);
```

Returns

uint32_t - the momentum falloff rate value.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetMomentumFalloffRate(laListWheelWidget* whl)

laListWheelWidget_GetRotationUpdateRate Function

Returns the wheel rotation update rate.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetRotationUpdateRate(laListWheelWidget* whl);
```

Returns

uint32_t - the rotation update rate value.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

uint32_t laListWheelWidget_GetRotationUpdateRate(laListWheelWidget* whl)

laListWheelWidget_GetShaded Function

Returns true if the list is using gradient shading to illustrate depth

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laBool laListWheelWidget_GetShaded(laListWheelWidget* whl);
```

Returns

laBool - true gradient shading is being used

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laBool laListWheelWidget_GetShaded(laListWheelWidget* whl)

laListWheelWidget_GetShowIndicators Function

Returns true if the list is displaying its selected item indicators

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laBool laListWheelWidget_GetShowIndicators(laListWheelWidget* whl);
```

Returns

laBool - true if the indicators are being shown

Parameters

Parameters	Description
laListWheelWidget* whl	the widget

Function

laBool laListWheelWidget_GetShowIndicators(laListWheelWidget* whl)

laListWheelWidget_GetVisibleItemCount Function

Returns the list's visible item count

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT uint32_t laListWheelWidget_GetVisibleItemCount(laListWheelWidget* whl);
```

Returns

uint32_t - the number of visible items

Parameters

Parameters	Description
laListWidget* lst	the widget

Function

uint32_t laListWheelWidget_GetVisibleItemCount(laListWheelWidget* whl)

laListWheelWidget_SetFlickInitSpeed Function

Configures the flick init speed for the list wheel

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetFlickInitSpeed(laListWheelWidget* whl, uint32_t speed);
```

Returns

laResult - the operation result

Description

The flick init speed is the drag distance needed to move the wheel into momentum mode. It is the distance that must be covered from one Aria update frame to another.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t speed	the flick init speed value

Function

laResult laListWheelWidget_SetFlickInitSpeed(laListWheelWidget* whl,
uint32_t speed)

laListWheelWidget_SetIndicatorArea Function

Configures the display area for the list selection indicator bars

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetIndicatorArea(laListWheelWidget* whl, uint32_t area);
```

Returns

laResult - the operation result

Description

This space is measured from the middle of the widget outward.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t area	the display area for the indicator bars

Function

laResult laListWheelWidget_SetIndicatorArea(laListWheelWidget* whl,
uint32_t area)

laListWheelWidget_SetMaxMomentum Function

Configures the maximum momentum value for the wheel

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetMaxMomentum(laListWheelWidget* whl, uint32_t max);
```

Returns

laResult - the operation result

Description

When a wheel is in momenum mode addition drag/flick gestures will add more momentum to the wheel. The maximum momentum value governs the maximum speed at which the wheel can rotate at any single point in time.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t max	the maximum momentum value

Function

laResult laListWheelWidget_SetMaxMomentum(laListWheelWidget* whl,
uint32_t max)

laListWheelWidget_SetMomentumFalloffRate Function

Configures the momentum falloff rate for the wheel

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetMomentumFalloffRate(laListWheelWidget* whl, uint32_t rate);
```

Returns

laResult - the operation result

Description

When a wheel is in momenum mode and during each rotation update tick the wheel will reduce its current momentum value by this falloff percentage. The higher this value is the faster a wheel will slow down. The wheel is limited to integer math so the lowest this value can be is one.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t max	the momentum falloff value

Function

laResult laListWheelWidget_SetMomentumFalloffRate(laListWheelWidget* whl,
uint32_t rate)

laListWheelWidget_SetRotationUpdateRate Function

Configures the rotation update rate for a wheel

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetRotationUpdateRate(laListWheelWidget* whl, uint32_t ms);
```

Returns

laResult - the operation result

Description

When a wheel is in momenum mode it may be too costly to update with every Aria update loop call. This value can delay a wheel update. For instance, if Aria is updating every 20ms, the wheel can be set to update every 60ms and it will update approximately every three to four Aria updates. This can cut down on the number of repaints the wheel needs to perform and can also slow the wheel down it if is rotating too fast for the application to handle. This value is typically expressed in milliseconds.

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
uint32_t ms	the desired rotation update rate

Function

laResult laListWheelWidget_SetRotationUpdateRate(laListWheelWidget* whl,
uint32_t ms)

laListWheelWidget_SetShaded Function

Configures the list to use gradient or flat background shading

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetShaded(laListWheelWidget* whl, laBool b);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
laBool b	true if gradient shading should be used

Function

laResult laListWheelWidget_SetShaded(laListWheelWidget* whl,
laBool b)

laListWheelWidget_SetShowIndicators Function

Configures the list to display the selected item indicator bars

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetShowIndicators(laListWheelWidget* whl, laBool b);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWheelWidget* whl	the widget
laBool b	configures the indicator bar display state

Function

laResult laListWheelWidget_SetShowIndicators(laListWheelWidget* whl,
laBool b)

laListWheelWidget_SetVisibleItemCount Function

Sets the number of visible items in the list. Must be greater than or equal to three and must be an odd number.

File

libaria_widget_listwheel.h

C

```
LIB_EXPORT laResult laListWheelWidget_SetVisibleItemCount(laListWheelWidget* whl, uint32_t cnt);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laListWidget* lst	the widget
uint32_t cnt	the desired number of items

Function

laResult laListWheelWidget_SetVisibleItemCount(laListWheelWidget* whl,
uint32_t cnt)

laRadioButtonWidget_SetImageMargin Function

Sets the distance between the icon and text

File

libaria_widget_radiobutton.h

C

```
LIB_EXPORT laResult laRadioButtonWidget_SetImageMargin(laRadioButtonWidget* btn, uint32_t mg);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRadioButtonWidget* btn	the widget
uint32_t mg	the distance value

Function

laResult laRadioButtonWidget_SetImageMargin(laRadioButtonWidget* btn,
uint32_t mg)

laScreen_GetMirrored Function

Returns the mirror setting for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laBool laScreen_GetMirrored(laScreen* scr);
```

Returns

laBool - the mirror setting

Description

Returns the mirror setting for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to reference

Function

laBool laScreen_GetMirrored(laScreen* scr)

laScreen_SetMirrored Function

Sets the mirror setting for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetMirrored(laScreen* scr, laBool mirr);
```

Returns

laResult - the result of the operation

Description

Sets the mirror setting for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to modify
laBool	the mirror setting

Function

laResult laScreen_SetMirrored(laScreen* scr, laBool mirr)

laString_Remove Function

Removes a number of characters from a string at a given index

File

libaria_string.h

C

```
LIB_EXPORT uint32_t laString_Remove(laString* str, uint32_t idx, uint32_t count);
```

Returns

uint32_t - the number of characters removed

Parameters

Parameters	Description
laString* str	the string to operate on
uint32_t idx	the index to remove from the number of characters to remove

Function

uint32_t laString_Remove(laString* str, uint32_t idx, uint32_t count)

laTextFieldWidget_SetClearOnFirstEdit Function

Sets the flag to indicate that the text field will be cleared on first edit.

File

libaria_widget_textfield.h

C

```
LIB_EXPORT laResult laTextFieldWidget_SetClearOnFirstEdit(laTextFieldWidget* txt, laBool clear);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laTextFieldWidget* txt	the widget
laBool clear	the desired flag state

Function

laResult laTextFieldWidget_SetClearOnFirstEdit(laTextFieldWidget* txt,
laBool clear)

laUtils_PickFromLayer Function

Finds the top-most visible widget in a layer at the given coordinates.

File

libaria_utils.h

C

```
LIB_EXPORT laWidget* laUtils_PickFromLayer(const laLayer* layer, int32_t x, int32_t y);
```

Returns

laWidget* - the result widget

Parameters

Parameters	Description
int32_t x	the x coordinate of the pick point
int32_t y	the y coordinate of the pick point

Function

laWidget* laUtils_PickFromLayer(const laLayer* layer, int32_t x, int32_t y)

laUtils_PointToLayerSpace Function

Converts a point from widget space into layer space

File

libaria_utils.h

C

```
void laUtils_PointToLayerSpace(laWidget* widget, GFX_Point* pnt);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Point* pnt	the point to convert

Function

void laUtils_PointToLayerSpace(laWidget* widget, GFX_Point* pnt)

laUtils_ScreenToMirroredSpace Function

Takes a point in screen space and returns a transformed version in mirrored space.

File

libaria_utils.h

C

```
GFX_Point laUtils_ScreenToMirroredSpace(const GFX_Point* pnt, const GFX_Rect* rect, GFX_Orientation ori);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Point* point	the point to transform
const GFX_Rect* rect	the screen dimensionrectangle
GFX_Orientation	the orientation setting

Function

void laUtils_ScreenToMirroredSpace(const GFX_Point* point,
const GFX_Rect* rect,
GFX_Orientation ori)

laUtils_ScreenToOrientedSpace Function

Takes a point in screen space and returns a transformed version in oriented space.

File

libaria_utils.h

C

```
GFX_Point laUtils_ScreenToOrientedSpace(const GFX_Point* pnt, const GFX_Rect* rect, GFX_Orientation ori);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Point* point	the point to transform
const GFX_Rect* rect	the screen dimensionrectangle
GFX_Orientation	the orientation setting

Function

void laUtils_ScreenToOrientedSpace(const GFX_Point* point,
const GFX_Rect* rect,
GFX_Orientation ori)

laUtils_WidgetLocalRect Function

Returns the bounding rectangle of a widget in local space

File

libaria_utils.h

C

```
GFX_Rect laUtils_WidgetLocalRect(laWidget* widget);
```

Returns

GFX_Rect - the bounding rectangle

Parameters

Parameters	Description
laWidget* widget	the subject widget

Function

GFX_Rect laUtils_WidgetLocalRect(laWidget* widget)

laWidget_GetBackgroundType Function

Return the property value 'background type' associated with the widget object

File

libaria_widget.h

C

```
LIB_EXPORT laBackgroundType laWidget_GetBackgroundType(laWidget* wgt);
```

Returns

laBackgroundType - the current background type

Description

Return the property value 'background type' associated with the widget object The background type property decides if the widget background is drawn and re-drawn. If background is none, the entire parent widget will be re-drawn in the event that the widget gets dirty and needs re-drawing.

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBackgroundType laWidget_GetBackgroundType(laWidget* wgt)

laWidget_GetOptimizationFlags Function

Returns the optimization flags for the widget

File

libaria_widget.h

C

```
LIB_EXPORT laBool laWidget_GetOptimizationFlags(laWidget* wgt);
```

Returns

laBool - the flag value

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laBool laWidget_GetOptimizationFlags(laWidget* wgt)

laWidget_SetBackgroundType Function

Set the property value 'background type' associated with the widget object

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetBackgroundType(laWidget* wgt, laBackgroundType type);
```

Returns

laResult - the operation result

Description

Set the property value 'draw background' associated with the widget object

Parameters

Parameters	Description
laWidget* wgt	the widget
laBackgroundType type	the desired background type

Function

laResult laWidget_SetBackgroundType(laWidget* wgt, laBackgroundType type)

laWidget_SetOptimizationFlags Function

Sets the optimizations for a widget

File

libaria_widget.h

C

```
LIB_EXPORT laResult laWidget_SetOptimizationFlags(laWidget* wgt, uint32_t flags);
```

Returns

laResult - the operation result

Description

See the optimizations enum for a descriptions of the individual flags

Parameters

Parameters	Description
laWidget* wgt	the widget

Function

laResult laWidget_SetOptimizationFlags(laWidget* wgt, uint32_t flags)

laRectangleWidget_SetThickness Function

Sets the rectangle border thickness setting

File

libaria_widget_rectangle.h

C

```
LIB_EXPORT laResult laRectangleWidget_SetThickness(laRectangleWidget* rect, int32_t thk);
```

Returns

laResult - the operation result

Parameters

Parameters	Description
laRectangleWidget* rect	the widget
int32_t thk	the thickness setting

Function

laResult laRectangleWidget_SetThickness(laRectangleWidget* rect,
int32_t thk)

laString_DrawClipped Function

Wrapper around GFX Utility string draw function for Aria user interface library. Draws only a clipped area of a string. Internal use only.

File

libaria_string.h

C

```
LIB_EXPORT void laString_DrawClipped(laString* str, int32_t strX, int32_t strY, int32_t strWidth, int32_t
strHeight, int32_t x, int32_t y, GFXU_ExternalAssetReader** reader);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to draw
int32_t strX	clipped x position
int32_t strY	clipped y position
int32_t strWidth	clipped rectangle width
int32_t strHeight	clipped rectangle height
int32_t x	x position to render at
int32_t y	y position to render at
GFXU_ExternalAssetReader** reader	external reader state machine, if string font is located external

Function

void laString_DrawClipped(laString* str,
int32_t strX,
int32_t strY,
int32_t strWidth,
int32_t strHeight,
int32_t x,
int32_t y,
GFXU_ExternalAssetReader** reader)

laString_IsEmpty Function

Returns a boolean indicating if the provided string contains data or has a link to the string table.

File

libaria_string.h

C

```
LIB_EXPORT laBool laString_IsEmpty(const laString* str);
```

Returns

laBool - LA_TRUE if the string has data, LA_FALSE if not

Parameters

Parameters	Description
const laString* str	the string to analyze

Function

laBool laString_IsEmpty(laString* str)

laUtils_GetNextHighestWidget Function

Gets the next highest Z order widget in the tree from 'wgt'

File

libaria_utils.h

C

```
laWidget* laUtils_GetNextHighestWidget(laWidget* wgt);
```

Returns

laWidget* - the next highest widget or NULL if 'wgt' is already the highest

Parameters

Parameters	Description
laWidget* wgt	the widget to analyze

Function

laBool laUtils_GetNextHighestWidget(laWidget* wgt)

laUtils_RectFromLayerSpace Function

Converts a rectangle from layer space to widget local space

File

libaria_utils.h

C

```
void laUtils_RectFromLayerSpace(laWidget* widget, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laWidget* widget	the subject widget
GFX_Rect* rect	the rectangle to convert

Function

void laUtils_RectFromLayerSpace(laWidget* widget, GFX_Rect* rect)

laUtils_WidgetIsOccluded Function

File

libaria_utils.h

C

```
laBool laUtils_WidgetIsOccluded(laWidget* wgt, const GFX_Rect* rect);
```

Description

This is function laUtils_WidgetIsOccluded.

laUtils_WidgetLayerRect Function

Returns the bounding rectangle of a widget in layer space

File

libaria_utils.h

C

```
GFX_Rect laUtils_WidgetLayerRect(laWidget* widget);
```

Returns

GFX_Rect - the bounding rectangle

Parameters

Parameters	Description
laWidget* widget	the subject widget

Function

GFX_Rect laUtils_WidgetLayerRect(laWidget* widget)

laWindowWidget_GetIconRect Function

File

libaria_widget_window.h

C

```
void laWindowWidget_GetIconRect(laWindowWidget* win, GFX_Rect* imgRect, GFX_Rect* imgSrcRect);
```

Description

This is function laWindowWidget_GetIconRect.

laWindowWidget_GetTextRect Function

File

libaria_widget_window.h

C

```
void laWindowWidget_GetTextRect(laWindowWidget* win, GFX_Rect* textRect, GFX_Rect* drawRect);
```

Description

This is function laWindowWidget_GetTextRect.

laWindowWidget_GetTitleBarRect Function

File

libaria_widget_window.h

C

```
void laWindowWidget_GetTitleBarRect(laWindowWidget* win, GFX_Rect* barRect);
```

Description

internal use only

laContext_IsDrawing Function

Indicates if any layers of the active screen are currently drawing a frame.

File

libaria_context.h

C

```
LIB_EXPORT laBool laContext_IsDrawing();
```

Returns

laResult

Description

Indicates if any layers are currently drawing a frame. Because frame updates can happen long after making changes to the UI state it is best to only make updates to the state of a layer tree only when the layer is not drawing.

Requires an active context and active screen.

Function

laBool laContext_IsDrawing()

laContext_IsLayerDrawing Function

Indicates if the layer at the given index of the active screen is currently drawing.

File

libaria_context.h

C

```
LIB_EXPORT laBool laContext_IsLayerDrawing(uint32_t idx);
```

Returns

laResult

Description

Indicates if the layer at the given index is currently drawing a frame. Because frame updates can happen long after making changes to the UI state it is best to only make updates to the state of a layer tree only when the layer is not drawing.

Requires an active context and active screen.

Parameters

Parameters	Description
uint32_t idx	the index of the layer to query

Function

laBool laContext_IsLayerDrawing(uint32_t idx)

laLayer_IsDrawing Function

Queries a layer to find out if it is currently drawing a frame.

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_IsDrawing(laLayer* layer);
```

Returns

laBool - the result of the operation

Parameters

Parameters	Description
laLayer* layer	the layer

Function

laBool laLayer_IsDrawing(laLayer* layer)

laLayer_GetInputRect Function

Gets the layer's input rectangle.

File

libaria_layer.h

C

```
LIB_EXPORT GFX_Rect laLayer_GetInputRect(const laLayer* layer);
```

Returns

GFX_Rect - the input rectangle

Description

Gets the layer's input rectangle.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

GFX_Rect laLayer_GetInputRect(const laLayer* layer)

laLayer_GetInputRectLocked Function

Gets the layer's input rect locked flag

File

libaria_layer.h

C

```
LIB_EXPORT laBool laLayer_GetInputRectLocked(const laLayer* layer);
```

Returns

laBool - the state of the layer's input rect locked flag

Description

This flag controls whether the layer input rectangle is locked to match the size of the layer's actual dimensions.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

laBool laLayer_GetInputRectLocked(const laLayer* layer)

laLayer_SetInputRect Function

Sets the layer's input rect dimensions.

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetInputRect(laLayer* layer, int32_t x, int32_t y, int32_t width, int32_t
height);
```

Returns

laResult - the result of the operation

Description

Sets the layer's input rect dimensions. This rectangle controls the input area of the layer. Some use cases may require a layer to accept input even if the input is outside of the physical dimensions of the layer. One example is a touch glass that is larger than the size of a display. Widgets may need to be placed in this invisible external area and still be capable of receiving input events.

Parameters

Parameters	Description
const laLayer* layer	the layer
int32_t x	the x position of the rectangle
int32_t y	the y position of the rectangle
int32_t width	the width of the rectangle
int32_t height	the height of the rectangle

Function

laResult laLayer_SetInputRect(laLayer* layer, laBool enable)

laLayer_SetInputRectLocked Function

Sets the layer's input rect locked flag.

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_SetInputRectLocked(laLayer* layer, laBool locked);
```

Returns

laResult - the result of the operation

Description

Sets the layer's input rect locked flag. This flag controls whether the input rectangle is locked to match the size of the layer's actual dimensions. When enabled, any change to the layer's size will be propagated to the input area as well. The default value is true.

Parameters

Parameters	Description
const laLayer* layer	the layer

Function

laResult laLayer_SetInputRectLocked(laLayer* layer, laBool enable)

laScreen_GetLayerSwapSync Function

Returns the layer swap sync setting for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laBool laScreen_GetLayerSwapSync(laScreen* scr);
```

Returns

laBool - the sync setting

Description

Returns the layer swap sync setting for the specified screen

Parameters

Parameters	Description
laScreen* scr	the screen to reference

Function

laBool laScreen_GetLayerSwapSync(laScreen* scr)

laScreen_SetLayerSwapSync Function

Sets the layer swap sync setting for the specified screen

File

libaria_screen.h

C

```
LIB_EXPORT laResult laScreen_SetLayerSwapSync(laScreen* scr, laBool sync);
```

Returns

laResult - the result of the operation

Description

Layer synchronization allows for the configuration timing of the buffer swap chain. In the case where multiple layers are being modified at the same time, it is often desirable to have the updates appear on the display at the same time. Layer sync will gate all layer swapping until all dirty layers have finished drawing. All layers will then swap at same time.

Parameters

Parameters	Description
laScreen* scr	the screen to modify
laBool	the sync setting

Function

laResult laScreen_SetLayerSwapSync(laScreen* scr, laBool sync)

laWidget_DeleteAllDescendants Function

Deletes all of the descendants of the given widget.

File

libaria_widget.h

C

```
LIB_EXPORT void laWidget_DeleteAllDescendants(laWidget* wgt);
```

Returns

void

Description

All descendants of this widget are removed and deleted.

Function

void laWidget_DeleteAllDescendants(laWidget* wgt)

laImageWidget_SetCallBackEnd Function

File

libaria_widget_image.h

C

```
void laImageWidget_SetCallBackEnd(laImageWidget* image, laImageWidget_DrawEventCallback cb);
```

Description

This is function laImageWidget_SetCallBackEnd.

laImageWidget_SetCallBackStart Function

File

libaria_widget_image.h

C

```
void laImageWidget_SetCallBackStart(laImageWidget* image, laImageWidget_DrawEventCallback cb);
```

Description

This is function laImageWidget_SetCallBackStart.

laString_DrawSubStringClipped Function

Wrapper around GFX Utility string draw function for Aria user interface library. Draws the substring between the start and end offset, and draws only the section of the string within the clipping rectangle. Internal use only.

File

libaria_string.h

C

```
LIB_EXPORT void laString_DrawSubStringClipped(laString* str, uint32_t start, uint32_t end, int32_t clipX,
int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t x, int32_t y, GFXU_ExternalAssetReader**
reader);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to draw
uint32_t start	the start position of the substring
uint32_t end	the end position of the substring
int32_t strX	clipped x position
int32_t strY	clipped y position
int32_t strWidth	clipped rectangle width
int32_t strHeight	clipped rectangle height
int32_t x	x position to render at
int32_t y	y position to render at

GFXU_ExternalAssetReader** reader	external reader state machine, if string font is located external

Function

void laString_DrawSubStringClipped(laString* str,
int32_t strX,
int32_t strY,
int32_t strWidth,
int32_t strHeight,
int32_t x,
int32_t y,
GFXU_ExternalAssetReader** reader)

laString_GetLineRect Function

Calculates the rectangle for a line in a string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The line ends when a line feed or end of string is reached.

File

libaria_string.h

C

```
LIB_EXPORT void laString_GetLineRect(laString* str, uint32_t start, GFX_Rect* rect, uint32_t * end);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to reference
uint32_t start	the start offset of the line in the string
GFX_Rect* rect	the calculated string rectangle result
uint32_t * end	the calculated end of the line (including line feed or end of string)

Function

void laString_GetLineRect(laString* str, uint32_t offset, GFX_Rect* rect, uint32_t * endoffset)

laString_GetMultiLineRect Function

Calculates the rectangle for a given multi-line string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The height the sum of the heights of the bounding rectangles for each line and the width is the widest among the bounding rectangles.

File

libaria_string.h

C

```
LIB_EXPORT void laString_GetMultiLineRect(laString* str, GFX_Rect* rect);
```

Returns

void

Parameters

Parameters	Description
laString* str	the string to reference
GFX_Rect* rect	the calculated string rectangle result

Function

void laString_GetMultiLineRect(laString* str, GFX_Rect* rect)

b) Data Types and Constants

laContext_t Structure

An instance of the Aria user interface library.

File

libaria_context.h

C

```
struct laContext_t {
  GFX_Display displayIndex;
  void* gfxContext;
  GFXU_MemoryIntf memIntf;
  laPreemptionLevel preemptLevel;
  laArray screenList;
  laScreen* activeScreen;
  uint32_t frameState;
  uint32_t currentLayer;
  laInputState input;
  GFXU_StringTableAsset* stringTable;
  uint32_t languageID;
  uint32_t widgetIDs;
  laScheme defaultScheme;
  GFX_ColorMode colorMode;
  laWidget* focus;
  laEditWidget* edit;
  laContext_ActiveScreenChangedCallback_FnPtr screenChangedCB;
  laContext_LanguageChangedCallback_FnPtr languageChangedCB;
};
```

Members

Members	Description
GFX_Display displayIndex;	the display the library is using
void* gfxContext;	the HAL context the library owns
GFXU_MemoryIntf memIntf;	the memory interface the library is using
laPreemptionLevel preemptLevel;	the preemption level the library is using
laArray screenList;	the list of the screens in the context
laScreen* activeScreen;	the currently active screen
uint32_t frameState;	the context frame state
uint32_t currentLayer;	the current drawing layer
laInputState input;	the input state of the instance
GFXU_StringTableAsset* stringTable;	the string table for the instance
uint32_t languageID;	the currently active language
uint32_t widgetIDs;	the next unique widget ID
laScheme defaultScheme;	an internal default scheme that widgets use by default if the user doesn't set one
GFX_ColorMode colorMode;	the color mode the library uses for all layers
laWidget* focus;	the widget that currently has focus
laEditWidget* edit;	the widget that is currently receiving edit events
laContext_ActiveScreenChangedCallback_FnPtr screenChangedCB;	screen changed callback
laContext_LanguageChangedCallback_FnPtr languageChangedCB;	language changed callback

Description

Structure: laContext

The context represents an discrete instance of Aria user interface library. The library is designed to be multi-instance and fully re-entrant. The entire state of the library is stored and referenced through the context pointer.

Remarks

None.

laList_t Structure

Linked list definition

File

libaria_list.h

C

```
struct laList_t {
  laListNode* head;
  laListNode* tail;
  size_t size;
};
```

Description

Structure: laList_t

Remarks

None.

laPreemptionLevel Enumeration

libaria pre-emption level values

File

libaria_common.h

C

```
enum laPreemptionLevel {
  LA_PREEMPTION_LEVEL_0,
  LA_PREEMPTION_LEVEL_1,
  LA_PREEMPTION_LEVEL_2
};
```

Members

Members	Description
LA_PREEMPTION_LEVEL_0	draw cycle always completes
LA_PREEMPTION_LEVEL_1	preempts after each widget fully draws
LA_PREEMPTION_LEVEL_2	preempts after each widget draw step completes

Description

Enumeration: laPreemptionLevel

libaria pre-emption level values

Remarks

None.

laRelativePosition Enumeration

libaria relative position values

File

libaria_common.h

C

```
enum laRelativePosition {
  LA_RELATIVE_POSITION_LEFTOF,
  LA_RELATIVE_POSITION_ABOVE,
  LA_RELATIVE_POSITION_RIGHTOF,
  LA_RELATIVE_POSITION_BELOW,
  LA_RELATIVE_POSITION_BEHIND
};
```

Description

Enumeration: laRelativePosition

libaria relative position values

Remarks

None.

GFXU_StringTableAsset Structure

Describes a string table asset. There is typically only ever one of these defined at any one time.

header - standard asset header languageCount - the number of languages in the string table stringCount - the number of strings in the string table stringIndexTable - the pointer to the string index table. the string index table is a table that contains all of the unique strings defined in the string table. fontTable - the font table contains an array of pointers to all defined font assets that the string table references fontIndexTable - the font index table is a table that maps strings to font indices which can then be used to get an actual font pointer from the font table encodingMode - indicates how strings are encoded in the stringIndexTable

File

gfxu_string.h

C

```
typedef struct GFXU_StringTableAsset_t {
  GFXU_AssetHeader header;
  uint32_t languageCount;
  uint32_t stringCount;
  uint8_t* stringIndexTable;
  GFXU_FontAsset** fontTable;
  uint8_t* fontIndexTable;
  GFXU_StringEncodingMode encodingMode;
} GFXU_StringTableAsset;
```

Description

Structure: GFXU_StringTableAsset_t

laBool Enumeration

libaria bool values

File

libaria_common.h

C

```
typedef enum laBool_t {
  LA_FALSE = 0,
  LA_TRUE
} laBool;
```

Description

Enumeration: laBool

libaria bool values

Remarks

None.

laContext Type

An instance of the Aria user interface library.

File

libaria_string.h

C

```
typedef struct laContext_t laContext;
```

Description

Structure: laContext

The context represents an discrete instance of Aria user interface library. The library is designed to be multi-instance and fully re-entrant. The entire state of the library is stored and referenced through the context pointer.

Remarks

None.

laContext_ActiveScreenChangedCallback_FnPtr Type

Callback pointer for the active screen change notification.

File

libaria_context.h

C

```
typedef void (* laContext_ActiveScreenChangedCallback_FnPtr)(int32_t, int32_t);
```

Description

Type: laContext_ActiveScreenChangedCallback_FnPtr

Callback pointer for the active screen change notification.

laContext_LanguageChangedCallback_FnPtr Type

Callback pointer for when the language change event occurs.

File

libaria_context.h

C

```
typedef void (* laContext_LanguageChangedCallback_FnPtr)(uint32_t);
```

Description

Type: laContext_LanguageChangedCallback_FnPtr

Callback pointer for when the language change event occurs.

laEditWidget Structure

Specifies the edit widget structure to manage all properties and events associated with edit widgets

File

libaria_editwidget.h

C

```
typedef struct laEditWidget_t {
  laWidget widget;
  laEditWidget_StartEdit_FnPtr startEdit;
  laEditWidget_EndEdit_FnPtr endEdit;
  laEditWidget_Clear_FnPtr clear;
  laEditWidget_Accept_FnPtr accept;
  laEditWidget_Set_FnPtr set;
  laEditWidget_Append_FnPtr append;
  laEditWidget_Backspace_FnPtr backspace;
} laEditWidget;
```

Description

Structure: laEditWidget_t

Edit widgets are a subset of normal widgets that are capable of receiving edit events from the UI kernel. Specialized widgets are capable of broadcasting edit events and the active edit event will react to them.

Remarks

None.

laEditWidget_Accept_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_Accept_FnPtr)(laEditWidget*);
```

Description

This is type laEditWidget_Accept_FnPtr.

laEditWidget_Append_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_Append_FnPtr)(laEditWidget*, laString);
```

Description

This is type laEditWidget_Append_FnPtr.

laEditWidget_Backspace_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_Backspace_FnPtr)(laEditWidget*);
```

Description

This is type laEditWidget_Backspace_FnPtr.

laEditWidget_Clear_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_Clear_FnPtr)(laEditWidget*);
```

Description

This is type laEditWidget_Clear_FnPtr.

laEditWidget_EndEdit_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_EndEdit_FnPtr)(laEditWidget*);
```

Description

This is type laEditWidget_EndEdit_FnPtr.

laEditWidget_Set_FnPtr Type

File

libaria_editwidget.h

C

```
typedef void (* laEditWidget_Set_FnPtr)(laEditWidget*, laString);
```

Description

This is type laEditWidget_Set_FnPtr.

laEditWidget_StartEdit_FnPtr Type

File

libaria_editwidget.h

C

```
typedef laResult (* laEditWidget_StartEdit_FnPtr)(laEditWidget*);
```

Description

This is type laEditWidget_StartEdit_FnPtr.

laHAlignment Enumeration

libaria horizontal alignment values

File

libaria_common.h

C

```
typedef enum {
  LA_HALIGN_LEFT,
  LA_HALIGN_CENTER,
  LA_HALIGN_RIGHT
} laHAlignment;
```

Description

Enumeration: laHAlignment

libaria horizontal alignment values

Remarks

None.

laList Type

Linked list definition

File

libaria_utils.h

C

```
typedef struct laList_t laList;
```

Description

Structure: laList_t

Remarks

None.

laListNode Structure

Linked list node definition

File

libaria_list.h

C

```
typedef struct laListNode_t {
  struct laListNode_t* next;
  void* val;
} laListNode;
```

Description

Structure: laListNode_t

Remarks

None.

laMargin Structure

libaria margin values

File

libaria_common.h

C

```
typedef struct laMargin_t {
  uint8_t left;
  uint8_t top;
  uint8_t right;
  uint8_t bottom;
} laMargin;
```

Description

Enumeration: laMargin

libaria margin values

Remarks

None.

laResult Enumeration

libaria results (success and failure codes).

File

libaria_common.h

C

```
typedef enum laResult_t {
  LA_FAILURE = -1,
  LA_SUCCESS = 0
} laResult;
```

Description

Enumeration: laResult

Various definitions for success and failure codes.

Remarks

None.

laScreen Structure

The structure to maintain the screen related variables and event handling

File

libaria_screen.h

C

```
typedef struct laScreen_t {
  uint32_t id;
  laString name;
  laBool persistent;
  laScreen_CreateCallback_FnPtr createCB;
  laBool created;
  laLayer* layers[LA_MAX_LAYERS];
  laScreenOrientation orientation;
  laBool mirrored;
  laBool layerSwapSync;
  laScreen_ShowHideCallback_FnPtr showCB;
  laScreen_ShowHideCallback_FnPtr hideCB;
} laScreen;
```

Members

Members	Description
uint32_t id;	the id of the screen
laString name;	the name of the screen
laBool persistent;	indicates that the screen should not free its widgets when it hides
laScreen_CreateCallback_FnPtr createCB;	the function that is called to create the contents of the screen
laBool created;	indicates if the screen currently exists
laLayer* layers[LA_MAX_LAYERS];	the layer array for the screen
laScreenOrientation orientation;	the orientation of the screen
laBool mirrored;	the mirror flag of the screen
laBool layerSwapSync;	the layerSwapSync flag of the screen
laScreen_ShowHideCallback_FnPtr showCB;	a callback that is called when the screen is shown
laScreen_ShowHideCallback_FnPtr hideCB;	a callback that is called when the screen is hidden

Description

Structure: laScreen_t

Maintains the layers associated with the screen. Marks the screen as persistent or not, which either destroys the screen when changed or preserves it for future reloading. Allocates and manages the event handling when screen change / show / hide events occur.

Remarks

None.

laString Structure

String definition

File

libaria_string.h

C

```
typedef struct laString_t {
  GFXU_CHAR* data;
  uint32_t capacity;
  uint32_t length;
  GFXU_FontAsset* font;
  int32_t table_index;
} laString;
```

Members

Members	Description
GFXU_CHAR* data;	local string data storage
uint32_t capacity;	actual memory capacity of the string
uint32_t length;	actual length of the string, typically this is capacity - 1, but can be less.
GFXU_FontAsset* font;	the font that contains the glyph raster data for this string
int32_t table_index;	if this is not LA_STRING_NULLIDX then this string is referencing an index in the string table. string table references are read-only but can be extracted to local modifiable versions

Description

Structure: laString_t

Remarks

None.

laVAlignment Enumeration

libaria vertical alignment values

File

libaria_common.h

C

```
typedef enum {
  LA_VALIGN_TOP,
  LA_VALIGN_MIDDLE,
  LA_VALIGN_BOTTOM
} laVAlignment;
```

Description

Enumeration: laVAlignment

libaria vertical alignment values

Remarks

None.

laButtonWidget_t Structure

Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.

File

libaria_widget_button.h

C

```
struct laButtonWidget_t {
  laWidget widget;
  laButtonState state;
  uint8_t toggleable;
  laString text;
  laHAlignment halign;
  laVAlignment valign;
  GFXU_ImageAsset* pressedImage;
  GFXU_ImageAsset* releasedImage;
  laRelativePosition imagePosition;
  uint32_t imageMargin;
  int32_t pressedOffset;
  laButtonWidget_PressedEvent pressedEvent;
  laButtonWidget_ReleasedEvent releasedEvent;
  GFXU_ExternalAssetReader* reader;
};
```

Members

Members	Description
laWidget widget;	base widget header
laButtonState state;	button state
uint8_t toggleable;	indicates if the button is toggleable
laString text;	the string that holds the button text
laHAlignment halign;	horizontal alignment of the button
laVAlignment valign;	vertical alignment of the button
GFXU_ImageAsset* pressedImage;	button pressed icon image
GFXU_ImageAsset* releasedImage;	button released icon image
laRelativePosition imagePosition;	icon position in relation to text
uint32_t imageMargin;	distance between text and icon
int32_t pressedOffset;	pressed text offset distance
laButtonWidget_PressedEvent pressedEvent;	pressed event callback
laButtonWidget_ReleasedEvent releasedEvent;	released event callback
GFXU_ExternalAssetReader* reader;	external asset reader state

Description

Structure: laButtonWidget_t

Remarks

None.

laLayer_t Structure

Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.

File

libaria_layer.h

C

```
struct laLayer_t {
  laWidget widget;
  laScreen* screen;
  laBool deleting;
  uint32_t bufferCount;
  laLayerBuffer buffers[GFX_MAX_BUFFER_COUNT];
  laBool alphaEnable;
  laBool maskEnable;
  GFX_Color maskColor;
  laBool vsync;
  laRectArray prevDamageRects;
  laRectArray currentDamageRects;
  laRectArray pendingDamageRects;
  laRectArray scratchRectList;
  laRectArray frameRectList;
  uint32_t frameRectIdx;
  GFX_Rect clippedDrawingRect;
  laBool drawingPrev;
  laLayerFrameState frameState;
  uint32_t layerDrawCount;
  uint32_t frameDrawCount;
  GFX_Rect inputRect;
  laBool inputRectLocked;
  laBool allowInputPassThrough;
  uint32_t deltaTime;
};
```

Members

Members	Description
laWidget widget;	base widget
laScreen* screen;	owning screen pointer

laBool deleting;	flag indicating that no changes should be made to the layer because it is in the process of being deleted
uint32_t bufferCount;	number of buffers in the layer
laLayerBuffer buffers[GFX_MAX_BUFFER_COUNT];	buffer array
laBool alphaEnable;	layer-based alpha blending enable flag
laBool maskEnable;	layer-based color masking enable flag
GFX_Color maskColor;	layer-based masking color value
laBool vsync;	layer vsync flag
laRectArray prevDamageRects;	previous damaged rectangle list
laRectArray currentDamageRects;	queued damaged rectangle list
laRectArray pendingDamageRects;	pending damaged rectangle list these are rectangles added during a frame in progress
laRectArray scratchRectList;	used for rectangle culling phase
laRectArray frameRectList;	this of rects to draw for a frame GFX_Rect currentDrawingRect; // the current damage rectangle
GFX_Rect clippedDrawingRect;	the current damage rectangle clipped to the currently rendering widget
laBool drawingPrev;	indicates if the layer is currently drawing from its previous rectangle array
laLayerFrameState frameState;	the current frame render state of the layer
uint32_t layerDrawCount;	the number of times this layer has drawn
uint32_t frameDrawCount;	the number of widgets that have rendered on this layer this frame
GFX_Rect inputRect;	layer input area
laBool inputRectLocked;	input area matches layer dimensions
laBool allowInputPassThrough;	indicates that input events should be propagated through the layer node to left siblings
uint32_t deltaTime;	stores delta time for updates that happen during rendering

Description

Structure: laLayer_t

Remarks

None.

laRadioButtonGroup_t Structure

Defines the structure used for the Radio Button group.

File

libaria_radiobutton_group.h

C

```
struct laRadioButtonGroup_t {
  laArray buttonList;
  laBool initialized;
  laRadioButtonWidget* selected;
};
```

Description

Structure laRadioButtonGroup_t

Defines the parameters required for a Radio Button group. Marks the current selected Radio button within the group

Remarks

None.

GFX_Point Type

A two dimensional Cartesian point.

File

libaria_utils.h

C

```
typedef struct GFX_Point_t GFX_Point;
```

Description

Structure: GFX_Point_t

GFX_Rect Type

A rectangle definition.

File

libaria_utils.h

C

```
typedef struct GFX_Rect_t GFX_Rect;
```

Description

Structure: GFX_Rect_t

laBorderType Enumeration

Specifies the different border types used for the widgets in the library

File

libaria_widget.h

C

```
typedef enum laBorderType_t {
  LA_WIDGET_BORDER_NONE,
  LA_WIDGET_BORDER_LINE,
  LA_WIDGET_BORDER_BEVEL,
  LA_WIDGET_BORDER_LAST = LA_WIDGET_BORDER_BEVEL
} laBorderType;
```

Description

Enumeration: laBorderType_t

Specifies the different border types used for the widgets in the library

Remarks

None.

laButtonState Enumeration

Controls the button pressed state

File

libaria_widget_button.h

C

```
typedef enum laButtonState_t {
  LA_BUTTON_STATE_UP,
  LA_BUTTON_STATE_DOWN,
  LA_BUTTON_STATE_TOGGLED
} laButtonState;
```

Description

Enumeration: laButtonState_t

laButtonWidget Type

Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.

File

libaria_widget_keypad.h

C

```
typedef struct laButtonWidget_t laButtonWidget;
```

Description

Structure: laButtonWidget_t

Remarks

None.

laButtonWidget_PressedEvent Type

File

libaria_widget_button.h

C

```
typedef void (* laButtonWidget_PressedEvent)(laButtonWidget*);
```

Description

This is type laButtonWidget_PressedEvent.

laButtonWidget_ReleasedEvent Type

File

libaria_widget_button.h

C

```
typedef void (* laButtonWidget_ReleasedEvent)(laButtonWidget*);
```

Description

This is type laButtonWidget_ReleasedEvent.

laCheckBoxWidget Structure

Implementation of a checkbox widget.

File

libaria_widget_checkbox.h

C

```
typedef struct laCheckBoxWidget_t {
  laWidget widget;
  laBool checked;
  laString text;
  laHAlignment halign;
  laVAlignment valign;
  GFXU_ImageAsset* checkedImage;
  GFXU_ImageAsset* uncheckedImage;
  laRelativePosition imagePosition;
  uint32_t imageMargin;
  GFXU_ExternalAssetReader* reader;
  laCheckBoxWidget_CheckedEvent checkedEvent;
  laCheckBoxWidget_CheckedEvent uncheckedEvent;
} laCheckBoxWidget;
```

Members

Members	Description
laWidget widget;	base class properties
laBool checked;	the state of the box
laString text;	the text of the box

laHAlignment halign;	the horizontal alignment of the box contents
laVAlignment valign;	the vertical alignment of the box contents
GFXU_ImageAsset* checkedImage;	pointer to a custom image to use for the checked image
GFXU_ImageAsset* uncheckedImage;	pointer to a custom image to use for the unchecked image
laRelativePosition imagePosition;	postition of the image relative to the text of the box
uint32_t imageMargin;	the distance between the image and the text
GFXU_ExternalAssetReader* reader;	an external asset reader pointer
laCheckBoxWidget_CheckedEvent checkedEvent;	callback for checked events
laCheckBoxWidget_CheckedEvent uncheckedEvent;	callback for unchecked events

Description

Structure: laCheckBoxWidget_t

A check box widget contains an interactive two-state box indicating on or off. The check box may also contain descriptive text. Custom images for the check box may be used in place of the default box graphic.

Remarks

None.

laCheckBoxWidget_CheckedEvent Type

File

libaria_widget_checkbox.h

C

```
typedef void (* laCheckBoxWidget_CheckedEvent)(laCheckBoxWidget*);
```

Description

This is type laCheckBoxWidget_CheckedEvent.

laCheckBoxWidget_UncheckedEvent Type

File

libaria_widget_checkbox.h

C

```
typedef void (* laCheckBoxWidget_UncheckedEvent)(laCheckBoxWidget*);
```

Description

This is type laCheckBoxWidget_UncheckedEvent.

laCircleWidget Structure

Implementation of a circle widget.

File

libaria_widget_circle.h

C

```
typedef struct laCircleWidget_t {
  laWidget widget;
  int32_t x;
  int32_t y;
  int32_t radius;
} laCircleWidget;
```

Members

Members	Description
laWidget widget;	base widget header

int32_t x;	the origin x cooridnate
int32_t y;	the origin y coordinate
int32_t radius;	the radius of the circle

Description

Structure: laCircleWidget_t

A circle widget draws a circle of the specified origin and radius inside the widget bounds. All coordinates are expressed in local widget space.

The color of the circle is determined by the widget scheme's 'foreground' color.

Remarks

None.

laDrawSurfaceWidget Structure

Implementation of a Drawsurface widget.

File

libaria_widget_drawsurface.h

C

```
typedef struct laDrawSurfaceWidget_t {
  laWidget widget;
  laDrawSurfaceWidget_DrawCallback cb;
} laDrawSurfaceWidget;
```

Members

Members	Description
laWidget widget;	the widget base class
laDrawSurfaceWidget_DrawCallback cb;	the draw callback

Description

Structure: laDrawSurfaceWidget_t

A draw surface widget is a special widget that allows an application to perform custom HAL draw calls during Aria's paint loop. To use, create and add a draw surface widget to the desired place in the widget tree. Then register for the callback through the API 'laDrawSurfaceWidget_SetDrawCallback'. This callback occurs during the paint loop. The application should then be free to adjust the HAL draw state and issue draw calls as desired. The HAL layer, buffer, or context state must not be adjusted in any way.

It is also important to not stall for too long during the draw callback.

Remarks

None.

laDrawSurfaceWidget_DrawCallback Type

File

libaria_widget_drawsurface.h

C

```
typedef laBool (* laDrawSurfaceWidget_DrawCallback)(laDrawSurfaceWidget* sfc, GFX_Rect* bounds);
```

Description

This is type laDrawSurfaceWidget_DrawCallback.

laEvent Structure

Basic UI event definition

File

libaria_event.h

C

```
typedef struct laEvent_t {
```

```
  laEventID id;
} laEvent;
```

Description

Structure: laEvent_t

laEvent_FilterEvent Type

Function pointer to define an event filter. Event filters allow a receiver to discard undesirable events

File

libaria_event.h

C

```
typedef laBool (* laEvent_FilterEvent)(laEvent*);
```

Description

Function Pointer: laEvent_FilterEvent

laEventID Enumeration

Defines internal event type IDs

File

libaria_event.h

C

```
typedef enum laEventID_t {
  LA_EVENT_NONE,
  LA_EVENT_SCREEN_CHANGE,
  LA_EVENT_TOUCH_DOWN,
  LA_EVENT_TOUCH_UP,
  LA_EVENT_TOUCH_MOVED
} laEventID;
```

Members

Members	Description
LA_EVENT_NONE	internal events

Description

Enumeration: laEventID

laEventState Structure

Structure to manage the event lists, state and call back pointers

File

libaria_event.h

C

```
typedef struct laEventState_t {
  OSAL_SEM_HANDLE_TYPE eventCountSem;
  OSAL_MUTEX_HANDLE_TYPE eventLock;
  laList events;
  laEvent_FilterEvent filter;
} laEventState;
```

Description

Structure: laEventState_t

Remarks

None.

laGestureID Enumeration

Placeholder for eventual gesture support.

File

libaria_input.h

C

```
typedef enum laGestureID_t {
  LA_GESTURE_NONE = 0
} laGestureID;
```

Description

Enumeration: laGestureID

Remarks

None.

laGradientWidget Structure

Gradient widget struct definition.

File

libaria_widget_gradient.h

C

```
typedef struct laGradientWidget_t {
  laWidget widget;
  laGradientWidgetDirection dir;
} laGradientWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laGradientWidgetDirection dir;	gradient direction

Description

Enumeration: laGradientWidget_t

Remarks

None.

laGradientWidgetDirection Enumeration

Implementation of a gradient widget.

File

libaria_widget_gradient.h

C

```
typedef enum laGradientWidgetDirection_t {
  LA_GRADIENT_DIRECTION_RIGHT,
  LA_GRADIENT_DIRECTION_DOWN,
  LA_GRADIENT_DIRECTION_LEFT,
  LA_GRADIENT_DIRECTION_UP
} laGradientWidgetDirection;
```

Description

Enumeration: laGradientWidgetDirection_t

A gradient widget is similar to a panel widget with the exception that it can draw a gradient color for its background. This operation can be more costly than drawing a solid color and should be used sparingly.

Gradient uses 'foreground' and 'foreground inactive' as its interpolated background draw colors.

Remarks

None.

laGroupBoxWidget Structure

Group box struct definition.

File

libaria_widget_groupbox.h

C

```
typedef struct laGroupBoxWidget_t {
  laWidget widget;
  laString text;
  laHAlignment halign;
  GFXU_ExternalAssetReader* reader;
} laGroupBoxWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laString text;	group box title text
laHAlignment halign;	group box text alignment
GFXU_ExternalAssetReader* reader;	asset reader

Description

Enumeration: laGroupBoxWidget_t

A group box is a widget that is similar to a basic panel but provides a line border and title text. Used for grouping and describing widgets of similar function.

Remarks

None.

laImageSequenceEntry Structure

Image sequence entry definition

File

libaria_widget_imagesequence.h

C

```
typedef struct laImageSequenceEntry_t {
  GFXU_ImageAsset* image;
  uint32_t delay;
  laHAlignment halign;
  laVAlignment valign;
} laImageSequenceEntry;
```

Members

Members	Description
GFXU_ImageAsset* image;	image asset pointer
uint32_t delay;	how many time units to display this entry
laHAlignment halign;	the horizontal alignment for this entry
laVAlignment valign;	the vertical alignment for this entry

Description

Enumeration: laImageSequenceEntry_t

Defines a single entry for the image sequence widget

Remarks

None.

laImageSequenceImageChangedEvent_FnPtr Type

File

libaria_widget_imagesequence.h

C

```
typedef void (* laImageSequenceImageChangedEvent_FnPtr)(laImageSequenceWidget*);
```

Description

This is type laImageSequenceImageChangedEvent_FnPtr.

laImageSequenceWidget Structure

Image sequence widget struct definition

File

libaria_widget_imagesequence.h

C

```
typedef struct laImageSequenceWidget_t {
  laWidget widget;
  uint32_t count;
  laImageSequenceEntry* images;
  int32_t activeIdx;
  laBool playing;
  uint32_t time;
  laBool repeat;
  laImageSequenceImageChangedEvent_FnPtr cb;
  GFXU_ExternalAssetReader* reader;
} laImageSequenceWidget;
```

Members

Members	Description
laWidget widget;	widget base class
uint32_t count;	number of image entries for this widget
laImageSequenceEntry* images;	image entry array
int32_t activeIdx;	currently displayed entry
laBool playing;	indicates that the widget is automatically cycling
uint32_t time;	current cycle time
laBool repeat;	indicates that the sequence should repeat when it reaches the end of the sequence
laImageSequenceImageChangedEvent_FnPtr cb;	callback when the image changes
GFXU_ExternalAssetReader* reader;	asset reader pointer

Description

Enumeration: laImageSequenceWidget_t

An image sequence widget is similar to an image widget with the additional capability of showing a sequence of images and automating the transition between them.

This widget is dependent on the time value provided to laUpdate. If laUpdate is not provided with time information this widget will not be able to automatically cycle.

Remarks

None.

laImageWidget Structure

Image widget struct definition

File

libaria_widget_image.h

C

```
typedef struct laImageWidget_t {
  laWidget widget;
  laHAlignment halign;
  laVAlignment valign;
  GFXU_ImageAsset* image;
  GFXU_ExternalAssetReader* reader;
  laImageWidget_DrawEventCallback ImageDrawStart;
  laImageWidget_DrawEventCallback ImageDrawEnd;
} laImageWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laHAlignment halign;	image horizontal alignment
laVAlignment valign;	image vertical alignment
GFXU_ImageAsset* image;	pointer to image asset
GFXU_ExternalAssetReader* reader;	asset reader

Description

Enumeration: laImageWidget_t

An image widget displays an image asset.

Remarks

None.

laInput_TouchDownEvent Structure

Register and handle the touch press detect event

File

libaria_input.h

C

```
typedef struct laInput_TouchDownEvent_t {
  int32_t touchID;
  int32_t x;
  int32_t y;
} laInput_TouchDownEvent;
```

Description

Structure: laInput_TouchDownEvent_t

Register and handle the touch press detect event

Remarks

None.

laInput_TouchMovedEvent Structure

Register and handle the touch coordinates changed event

File

libaria_input.h

C

```
typedef struct laInput_TouchMovedEvent_t {
  int32_t touchID;
  int32_t prevX;
  int32_t prevY;
  int32_t x;
  int32_t y;
} laInput_TouchMovedEvent;
```

Description

Structure: laInput_TouchMovedEvent_t

Register and handle the touch coordinates changed event

Remarks

None.

laInput_TouchUpEvent Structure

Register and handle the touch release detect event

File

libaria_input.h

C

```
typedef struct laInput_TouchUpEvent_t {
  int32_t touchID;
  int32_t x;
  int32_t y;
} laInput_TouchUpEvent;
```

Description

Structure: laInput_TouchUpEvent_t

Register and handle the touch release detect event

Remarks

None.

laInputState Structure

Maintain a history of touch states; currently libaria keeps track of the last touch state only.

File

libaria_input.h

C

```
typedef struct laInputState_t {
  laBool enabled;
  laTouchState touch[LA_MAX_TOUCH_STATES];
} laInputState;
```

Description

Structure: laInputState_t

Maintain a history of touch states; currently libaria keeps track of the last touch state only.

Remarks

None.

laKey Enumeration

All values possible for key entry from the libaria keyboard widget

File

libaria_input.h

C

```
typedef enum laKey_t {
  KEY_NULL = 0,
  KEY_ESCAPE,
  KEY_F1,
  KEY_F2,
  KEY_F3,
  KEY_F4,
```

```
    KEY_F5,
    KEY_F6,
    KEY_F7,
    KEY_F8,
    KEY_F9,
    KEY_F10,
    KEY_F11,
    KEY_F12,
    KEY_PRINTSCREEN,
    KEY_SCROLLLOCK,
    KEY_PAUSE,
    KEY_1,
    KEY_2,
    KEY_3,
    KEY_4,
    KEY_5,
    KEY_6,
    KEY_7,
    KEY_8,
    KEY_9,
    KEY_0,
    KEY_BACKQUOTE,
    KEY_TAB,
    KEY_CAPSLOCK,
    KEY_BRACKET_LEFT,
    KEY_BRACKET_RIGHT,
    KEY_SLASH,
    KEY_SEMICOLON,
    KEY_QUOTE,
    KEY_BACKSLASH,
    KEY_EQUALS,
    KEY_BACKSPACE,
    KEY_MINUS,
    KEY_COMMA,
    KEY_ENTER,
    KEY_PERIOD,
    KEY_A,
    KEY_B,
    KEY_C,
    KEY_D,
    KEY_E,
    KEY_F,
    KEY_G,
    KEY_H,
    KEY_I,
    KEY_J,
    KEY_K,
    KEY_L,
    KEY_M,
    KEY_N,
    KEY_O,
    KEY_P,
    KEY_Q,
    KEY_R,
    KEY_S,
    KEY_T,
    KEY_U,
    KEY_V,
    KEY_W,
    KEY_X,
    KEY_Y,
    KEY_Z,
    KEY_SPACE,
    KEY_LCTRL,
    KEY_RCTRL,
    KEY_LSHIFT,
    KEY_RSHIFT,
    KEY_LALT,
    KEY_RALT,
    KEY_LMETA,
    KEY_RMETA,
    KEY_INSERT,
    KEY_HOME,
    KEY_PAGEUP,
    KEY_END,
```

```
  KEY_PAGEDOWN,
  KEY_RIGHT,
  KEY_LEFT,
  KEY_DOWN,
  KEY_UP,
  KEY_NUMLOCK,
  KEY_KP_DIVIDE,
  KEY_KP_MULTIPLY,
  KEY_KP_MINUS,
  KEY_KP_PLUS,
  KEY_KP_ENTER,
  KEY_KP_1,
  KEY_KP_2,
  KEY_KP_3,
  KEY_KP_4,
  KEY_KP_5,
  KEY_KP_6,
  KEY_KP_7,
  KEY_KP_8,
  KEY_KP_9,
  KEY_KP_0,
  KEY_KP_PERIOD,
  KEY_LAST = KEY_KP_PERIOD
} laKey;
```

Description

Enumeration: laKey

All values possible for key entry from the libaria keyboard widget

Remarks

None.

laKeyPadCell Structure

Defines a key pad cell struct

File

libaria_widget_keypad.h

C

```
typedef struct laKeyPadCell_t {
  laBool enabled;
  laButtonWidget* button;
  laKeyPadCellAction action;
  laString value;
} laKeyPadCell;
```

Members

Members	Description
laBool enabled;	indicates if the cell should be drawn
laButtonWidget* button;	the button that handles the cell input events and rendering
laKeyPadCellAction action;	the action that occurs when the cell is activated
laString value;	the value that is passed to the edit event system

Description

Structure: laKeyPadCell_t

A key pad is made up of an array of key pad cells. Each cell is individually an laButtonWidget, an action, a value, and a few other options.

Remarks

None.

laKeyPadCellAction Enumeration

Defines an assigned action to a key pad cell

File

libaria_widget_keypad.h

C

```
typedef enum laKeyPadCellAction_t {
  LA_KEYPAD_CELL_ACTION_NONE,
  LA_KEYPAD_CELL_ACTION_APPEND,
  LA_KEYPAD_CELL_ACTION_SET,
  LA_KEYPAD_CELL_ACTION_BACKSPACE,
  LA_KEYPAD_CELL_ACTION_CLEAR,
  LA_KEYPAD_CELL_ACTION_ACCEPT
} laKeyPadCellAction;
```

Description

Structure: laKeyPadCellAction_t

Remarks

None.

laKeyPadWidget Structure

Defines a key pad widget struct

File

libaria_widget_keypad.h

C

```
typedef struct laKeyPadWidget_t {
  laWidget widget;
  uint32_t rows;
  uint32_t cols;
  laKeyPadActionTrigger trigger;
  laKeyPadCell* cells;
  laKeyPadWidget_KeyClickEvent clickEvt;
  GFXU_ExternalAssetReader* reader;
} laKeyPadWidget;
```

Members

Members	Description
laWidget widget;	widget base class
uint32_t rows;	number of button rows
uint32_t cols;	number of button columns
laKeyPadActionTrigger trigger;	trigger for action and events
laKeyPadCell* cells;	key cell array
laKeyPadWidget_KeyClickEvent clickEvt;	key click callback event
GFXU_ExternalAssetReader* reader;	asset reader

Description

Structure: laKeyPadCell_t

A key pad is a widget that is comprised of an array of laButtonWidgets. This widget serves to issue edit events based on application or input interaction. Receptor edit widgets can then receive these edit events and react accordingly.

Remarks

None.

laKeyPadWidget_KeyClickEvent Type

File

libaria_widget_keypad.h

C

```
typedef void (* laKeyPadWidget_KeyClickEvent)(laKeyPadWidget*, laButtonWidget*, uint32_t, uint32_t);
```

Description

This is type laKeyPadWidget_KeyClickEvent.

laLabelWidget Structure

Implementation of a label widget struct

File

libaria_widget_label.h

C

```
typedef struct laLabelWidget_t {
  laWidget widget;
  laString text;
  laHAlignment halign;
  laVAlignment valign;
  GFXU_ExternalAssetReader* reader;
} laLabelWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laString text;	string to draw
laHAlignment halign;	horizontal alignment of string
laVAlignment valign;	vertical alignment of string
GFXU_ExternalAssetReader* reader;	asset reader

Description

Structure: laLabelWidget_t

A label widget is a simple widget that draws a string of text.

Remarks

None.

laLayer Type

Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.

File

libaria_utils.h

C

```
typedef struct laLayer_t laLayer;
```

Description

Structure: laLayer_t

Remarks

None.

laLayerBuffer Structure

Structure to maintain the buffer type and track the buffer location for each layer

File

libaria_layer.h

C

```
typedef struct laLayerBuffer_t {
  laLayerBufferType type;
  void* address;
} laLayerBuffer;
```

Description

Structure: laLayerBuffer_t

Structure to maintain the buffer type and track the buffer location for each layer

Remarks

None.

laLayerBufferType Enumeration

Defines the type of a layer. If the layer has an explicit address then Aria tries to set that through the HAL when the layer is being set up.

File

libaria_layer.h

C

```
typedef enum laLayerBufferType_t {
  LA_BUFFER_TYPE_AUTO,
  LA_BUFFER_TYPE_ADDRESS
} laLayerBufferType;
```

Description

Enumeration: laLayerBufferType_t

Remarks

None.

laLineWidget Structure

Defines the implementation of a line widget struct

File

libaria_widget_line.h

C

```
typedef struct laLineWidget_t {
  laWidget widget;
  int32_t x1;
  int32_t y1;
  int32_t x2;
  int32_t y2;
} laLineWidget;
```

Members

Members	Description
laWidget widget;	widget base class
int32_t x1;	point 1 x
int32_t y1;	point 1 y
int32_t x2;	point 2 x
int32_t y2;	point 2 y

Description

Structure: laLineWidget_t

A line widget draws a simple line shape within the confines of its bounding rectangle. All coordinates are expressed in local widget space.

The color of the line is determined by the widget scheme's 'foreground' color.

Remarks

None.

laListItem Structure

Defines a list item struct

File

libaria_widget_list.h

C

```
typedef struct laListItem_t {
  laString string;
  GFXU_ImageAsset* icon;
  laBool selected;
  GFX_Rect rowRect;
} laListItem;
```

Members

Members	Description
laString string;	list item string
GFXU_ImageAsset* icon;	list item icon
laBool selected;	list item selected flag
GFX_Rect rowRect;	list item row rectangle

Description

Structure: laListItem_t

Remarks

None.

laListWheelItem Structure

Implementation of a list wheel widget item struct

File

libaria_widget_listwheel.h

C

```
typedef struct laListWheelItem_t {
  laString string;
  GFXU_ImageAsset* icon;
} laListWheelItem;
```

Description

Structure: laListWheelItem_t

A list wheel item contains either a text string, an icon, or both

Remarks

None.

laListWheelWidget Structure

Implementation of a list wheel widget struct

File

libaria_widget_listwheel.h

C

```
typedef struct laListWheelWidget_t {
  laWidget widget;
  laArray items;
  int32_t selectedItem;
  int32_t visibleItems;
  int32_t topItem;
  laHAlignment halign;
  laRelativePosition iconPos;
  uint32_t iconMargin;
  laBool showIndicators;
  uint32_t indicatorArea;
  uint32_t shaded;
```

```
  int32_t cycleDistance;
  int32_t cycleDelta;
  int32_t firstTouchY;
  int32_t touchY;
  int32_t lastTouchY;
  laBool stillTouching;
  int32_t minFlickDelta;
  int32_t momentum;
  int32_t maxMomentum;
  int32_t momentumFalloff;
  int32_t rotation;
  int32_t rotationCounter;
  int32_t rotationTick;
  laBool snapPending;
  laListWheelIndicatorFill indicatorFill;
  laListWheelZoomEffects zoomEffects;
  laBool autoHideWheel;
  laBool hideWheel;
  struct {
    int32_t y;
    int32_t per;
    uint32_t nextItem;
  } paintState;
  laListWheelWidget_SelectedItemChangedEvent cb;
  laBorderType borderTypeCache;
  laBackgroundType backgroundTypeCache;
  GFXU_ExternalAssetReader* reader;
} laListWheelWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laArray items;	list of items for the wheel
int32_t selectedItem;	currently selected item
int32_t visibleItems;	number of visible items in the wheel must be odd and >= 3
int32_t topItem;	the current top item
laHAlignment halign;	the horizontal alignment of the items
laRelativePosition iconPos;	the icon position of the items
uint32_t iconMargin;	the icon margin of the items
laBool showIndicators;	controls the visibility of the horizontal indicator bars in the center of the widget
uint32_t indicatorArea;	controls the distance between the indicator bars
uint32_t shaded;	determins if the background of the widget uses gradient shading to show depth
int32_t cycleDistance;	determins the amount of drag distance needed to cycle between items
int32_t cycleDelta;	tracks the current amount of drag distance
int32_t firstTouchY;	these track drag movement over time
int32_t minFlickDelta;	amount of distance that must be dragged in a single frame to trigger momentum mode
int32_t momentum;	current momentum value
int32_t maxMomentum;	maximum momentum value
int32_t momentumFalloff;	momentum falloff rate
int32_t rotation;	determines actual rotation of the wheel
int32_t rotationCounter;	time-based limiter for rotation calculations
int32_t rotationTick;	rotation time accumulator
laListWheelIndicatorFill indicatorFill;	the indicator's fill type
laListWheelZoomEffects zoomEffects;	zoomEffects
laBool autoHideWheel;	auto hides the wheel
laBool hideWheel;	flag to hide/show the wheel
laListWheelWidget_SelectedItemChangedEvent cb;	item changed callback
laBorderType borderTypeCache;	Copy of border type, used to restore borders on auto-hide
laBackgroundType backgroundTypeCache;	Copy of background type, used to restore borders on auto-hide
GFXU_ExternalAssetReader* reader;	asset reader

Description

Structure: laListWheelWidget_t

A list wheel widget is a widget that is similar to a normal list widget but can be dragged up or down to cycle through a single active value. This widget is also capable of momentum and motion over time.

Remarks

None.

laListWheelWidget_SelectedItemChangedEvent Type

File

libaria_widget_listwheel.h

C

```
typedef void (* laListWheelWidget_SelectedItemChangedEvent)(laListWheelWidget*, uint32_t idx);
```

Description

This is type laListWheelWidget_SelectedItemChangedEvent.

laListWidget Structure

Defines the implementation of a list widget

File

libaria_widget_list.h

C

```
typedef struct laListWidget_t {
  laWidget widget;
  laListWidget_SelectionMode mode;
  laBool allowEmpty;
  laArray items;
  laHAlignment halign;
  laRelativePosition iconPos;
  uint32_t iconMargin;
  uint32_t itemDown;
  laScrollBarWidget* scrollbar;
  struct {
    laListItem* item;
    GFX_Rect itemRect;
    int32_t y;
    uint32_t nextItem;
  } paintState;
  laListWidget_ItemSelectedChangedEvent cb;
  GFXU_ExternalAssetReader* reader;
} laListWidget;
```

Members

Members	Description
laWidget widget;	list base class
laListWidget_SelectionMode mode;	list selection mode
laBool allowEmpty;	indicates if the list must always have at least one selected item
laArray items;	list containing the list items
laHAlignment halign;	horizontal alignment of the list
laRelativePosition iconPos;	icon position for the list icons
uint32_t iconMargin;	margin for the list icons
uint32_t itemDown;	tracks whether an input event is in process
laScrollBarWidget* scrollbar;	internal scrollbar for this widget
laListWidget_ItemSelectedChangedEvent cb;	item selected changed event

Description

Structure: laListWidget_t

A list widget is a widget that contains a series of vertical nodes. Each node can have text, an image, or both, and can be selected or not. The list has a built-in scrollbar. This allows the list to be larger than the visible area of the widget.

Remarks

None.

laListWidget_ItemSelectedChangedEvent Type

File

libaria_widget_list.h

C

```
typedef void (* laListWidget_ItemSelectedChangedEvent)(laListWidget*, uint32_t idx, laBool selected);
```

Description

This is type laListWidget_ItemSelectedChangedEvent.

laListWidget_SelectedItemChangedEvent Type

File

libaria_widget_list.h

C

```
typedef void (* laListWidget_SelectedItemChangedEvent)(laListWidget*, uint32_t idx, laBool selected);
```

Description

This is type laListWidget_SelectedItemChangedEvent.

laListWidget_SelectionMode Enumeration

Defines the list selection modes

File

libaria_widget_list.h

C

```
typedef enum laListWidget_SelectionMode_t {
  LA_LIST_WIDGET_SELECTION_MODE_SINGLE,
  LA_LIST_WIDGET_SELECTION_MODE_MULTIPLE,
  LA_LIST_WIDGET_SELECTION_MODE_CONTIGUOUS
} laListWidget_SelectionMode;
```

Description

Enumeration: laListWidget_SelectionMode_t

Single - a single selection from the list is allowed at any one time Multiple - any number of selected items is allowed at any one time Contiguous - any number of selected items in a contiguous serious is allowed at any one time

Remarks

None.

laMouseButton Enumeration

All values possible for mouse key entry from the libaria mouse input

File

libaria_input.h

C

```
typedef enum laMouseButton_t {
  BUTTON_NONE = 0,
  BUTTON_LEFT,
  BUTTON_MIDDLE,
  BUTTON_RIGHT,
```

```
  BUTTON_WHEEL_UP,
  BUTTON_WHEEL_DOWN,
  BUTTON_LAST = BUTTON_WHEEL_DOWN
} laMouseButton;
```

Description

Enumeration: laMouseButton

All values possible for mouse key entry from the libaria mouse input

Remarks

None.

laProgressBar Type

File

libaria_widget_progressbar.h

C

```
typedef struct laProgressBarWidget_t laProgressBar;
```

Description

This is type laProgressBar.

laProgressBar_ValueChangedEventCallback Type

File

libaria_widget_progressbar.h

C

```
typedef void (* laProgressBar_ValueChangedEventCallback)(laProgressBar*, uint32_t);
```

Description

This is type laProgressBar_ValueChangedEventCallback.

laProgressBarDirection Enumeration

Defines the valid values for the progress bar widget fill directions.

File

libaria_widget_progressbar.h

C

```
typedef enum laProgressBarDirection_t {
  LA_PROGRESSBAR_DIRECTION_RIGHT,
  LA_PROGRESSBAR_DIRECTION_UP,
  LA_PROGRESSBAR_DIRECTION_LEFT,
  LA_PROGRESSBAR_DIRECTION_DOWN
} laProgressBarDirection;
```

Description

Enumeration: laProgressBarDirection_t

Remarks

None.

laProgressBarWidget Structure

Implementation of a progressbar widget struct

File

libaria_widget_progressbar.h

C

```
typedef struct laProgressBarWidget_t {
  laWidget widget;
  laProgressBarDirection direction;
  uint32_t value;
  laProgressBar_ValueChangedEventCallback cb;
} laProgressBarWidget;
```

Members

Members	Description
laWidget widget;	base widget class
laProgressBarDirection direction;	the fill direction of the bar
uint32_t value;	fill percentage
laProgressBar_ValueChangedEventCallback cb;	value changed callback

Description

Structure: laProgressBarDirection_t

A progress bar widget is a widget that can fill itself with a color based on a given percentage from 0-100. This is often used to visually illustrate the progress of some other activity over time.

Remarks

None.

laRadioButtonGroup Type

Defines the structure used for the Radio Button group.

File

libaria_widget_radiobutton.h

C

```
typedef struct laRadioButtonGroup_t laRadioButtonGroup;
```

Description

Structure laRadioButtonGroup_t

Defines the parameters required for a Radio Button group. Marks the current selected Radio button within the group

Remarks

None.

laRadioButtonWidget Structure

Implementation of a radio button widget struct

File

libaria_widget_radiobutton.h

C

```
typedef struct laRadioButtonWidget_t {
  laWidget widget;
  laBool selected;
  laString text;
  laHAlignment halign;
  laVAlignment valign;
  GFXU_ImageAsset* selectedImage;
  GFXU_ImageAsset* unselectedImage;
  laRelativePosition imagePosition;
  uint32_t imageMargin;
  uint32_t circleButtonSize;
  laRadioButtonWidget_SelectedEvent selectedEvent;
  laRadioButtonWidget_DeselectedEvent deselectedEvent;
  struct {
    uint8_t enabled;
  } paintData;
```

```
  GFXU_ExternalAssetReader* reader;
  laRadioButtonGroup* group;
} laRadioButtonWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laBool selected;	indicates if the radio button is selected
laString text;	radio button text
laHAlignment halign;	horizontal alignment
laVAlignment valign;	vertical alignment
GFXU_ImageAsset* selectedImage;	button custom selected image
GFXU_ImageAsset* unselectedImage;	buton custom unselected image
laRelativePosition imagePosition;	image icon relative position
uint32_t imageMargin;	image margin
uint32_t circleButtonSize;	size of radio circle button in pixels
laRadioButtonWidget_SelectedEvent selectedEvent;	button selected event callback
laRadioButtonWidget_DeselectedEvent deselectedEvent;	button deselected event callback
GFXU_ExternalAssetReader* reader;	asset reader
laRadioButtonGroup* group;	radio button group

Description

Enumeration: laRadioButtonWidget_t

A radio button is similar to a checkbox widget in that it has an on and off state. It is further capable of being added to a radio button group. This group provides a mutually exclusive selection capability so that only one radio button may be selected at any one time.

Remarks

None.

laRadioButtonWidget_DeselectedEvent Type

File

libaria_widget_radiobutton.h

C

```
typedef void (* laRadioButtonWidget_DeselectedEvent)(laRadioButtonWidget*);
```

Description

This is type laRadioButtonWidget_DeselectedEvent.

laRadioButtonWidget_SelectedEvent Type

File

libaria_widget_radiobutton.h

C

```
typedef void (* laRadioButtonWidget_SelectedEvent)(laRadioButtonWidget*);
```

Description

This is type laRadioButtonWidget_SelectedEvent.

laRectangleWidget Structure

Implementation of a rectangle widget struct

File

libaria_widget_rectangle.h

C

```
typedef struct laRectangleWidget_t {
  laWidget widget;
  int32_t thickness;
} laRectangleWidget;
```

Members

Members	Description
laWidget widget;	widget base class
int32_t thickness;	rectangle border thickness

Description

Enumeration: laRectangleWidget_t

A rectangle widget draws a basic rectangle of a given thickness using the widget's bounding box as the dimensions.

Remarks

None.

laScheme Structure

This structure specifies the style scheme components of an object.

File

libaria_scheme.h

C

```
typedef struct laScheme_t {
  GFX_Color base;
  GFX_Color highlight;
  GFX_Color highlightLight;
  GFX_Color shadow;
  GFX_Color shadowDark;
  GFX_Color foreground;
  GFX_Color foregroundInactive;
  GFX_Color foregroundDisabled;
  GFX_Color background;
  GFX_Color backgroundInactive;
  GFX_Color backgroundDisabled;
  GFX_Color text;
  GFX_Color textHighlight;
  GFX_Color textHighlightText;
  GFX_Color textInactive;
  GFX_Color textDisabled;
} laScheme;
```

Description

Enumeration: laScheme_t

A scheme is a collection of colors that can be referenced by widgets or other objects. While the color names strive to be intuitive they aren't always used in the manner in which they describe.

Remarks

None.

laScreen_CreateCallback_FnPtr Type

Callback pointer for a new screen create event notification. This is called when the library attempts to create a screen.

File

libaria_screen.h

C

```
typedef void (* laScreen_CreateCallback_FnPtr)(laScreen*);
```

Description

Type: laScreen_CreateCallback_FnPtr

laScreen_ShowHideCallback_FnPtr Type

Callback pointer for the active screen show or hide event change notification.

File

libaria_screen.h

C

```
typedef void (* laScreen_ShowHideCallback_FnPtr)(laScreen*);
```

Description

Type: laScreen_ShowHideCallback_FnPtr

Callback pointer for the active screen show or hide event change notification.

laScreenOrientation Enumeration

Possible values for screen orientation.

File

libaria_screen.h

C

```
typedef enum laScreenOrientation_t {
  LA_SCREEN_ORIENTATION_0 = 0x0,
  LA_SCREEN_ORIENTATION_90,
  LA_SCREEN_ORIENTATION_180,
  LA_SCREEN_ORIENTATION_270
} laScreenOrientation;
```

Description

Enumeration: laScreenOrientation_t

Possible values for screen orientation.

Remarks

None.

laScrollBarOrientation Enumeration

Defines the scroll bar direction values

File

libaria_widget_scrollbar.h

C

```
typedef enum laScrollBarOrientation_t {
  LA_SCROLLBAR_ORIENT_VERTICAL,
  LA_SCROLLBAR_ORIENT_HORIZONTAL
} laScrollBarOrientation;
```

Description

Enumeration: laScrollBarOrientation_t

Remarks

None.

laScrollBarState Enumeration

Defines the various scroll bar state values

File

libaria_widget_scrollbar.h

C

```
typedef enum laScrollBarState_t {
  LA_SCROLLBAR_STATE_NONE,
  LA_SCROLLBAR_STATE_TOP_PRESSED,
  LA_SCROLLBAR_STATE_TOP_INSIDE,
  LA_SCROLLBAR_STATE_BOTTOM_PRESSED,
  LA_SCROLLBAR_STATE_BOTTOM_INSIDE,
  LA_SCROLLBAR_STATE_HANDLE_DOWN
} laScrollBarState;
```

Description

Enumeration: laScrollBarState_t

Remarks

None.

laScrollBarWidget Structure

Implementation of a scroll bar widget.

File

libaria_widget_scrollbar.h

C

```
typedef struct laScrollBarWidget_t {
  laWidget widget;
  laScrollBarState state;
  laScrollBarOrientation alignment;
  uint32_t max;
  uint32_t extent;
  uint32_t value;
  uint32_t step;
  laScrollBarWidget_ValueChangedEvent valueChangedEvent;
  GFX_Point handleDownOffset;
} laScrollBarWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laScrollBarState state;	scrollbar input state
laScrollBarOrientation alignment;	scroll bar direction
uint32_t max;	maximum scroll value
uint32_t extent;	visible space/handle size
uint32_t value;	current scroll value
uint32_t step;	discreet scroll stepping value
laScrollBarWidget_ValueChangedEvent valueChangedEvent;	value changed callback

Description

Structure: laScrollBarWidget_t

A scroll bar is a widget that is capable of displaying a range and a scroll handle. The handle can grow and shrink in size depending on the scroll range and visible scroll space and can be interacted with to scroll through the available space.

Remarks

None.

laScrollBarWidget_ValueChangedEvent Type

File

libaria_widget_scrollbar.h

C

```
typedef void (* laScrollBarWidget_ValueChangedEvent)(laScrollBarWidget*);
```

Description

This is type laScrollBarWidget_ValueChangedEvent.

laSliderOrientation Enumeration

Slider orientations

File

libaria_widget_slider.h

C

```
typedef enum laSliderOrientation_t {
  LA_SLIDER_ORIENT_VERTICAL,
  LA_SLIDER_ORIENT_HORIZONTAL
} laSliderOrientation;
```

Description

Enumeration: laSliderOrientation_t

Remarks

None.

laSliderState Enumeration

Describes various slider states

File

libaria_widget_slider.h

C

```
typedef enum laSliderState_t {
  LA_SLIDER_STATE_NONE,
  LA_SLIDER_STATE_HANDLE_DOWN,
  LA_SLIDER_STATE_AREA_DOWN
} laSliderState;
```

Description

Enumeration: laSliderState_t

Remarks

None.

laSliderWidget Structure

Implementation of a slider widget struct

File

libaria_widget_slider.h

C

```
typedef struct laSliderWidget_t {
  laWidget widget;
  laSliderState state;
  laSliderOrientation alignment;
  int32_t min;
```

```
  int32_t max;
  int32_t value;
  uint32_t grip;
  laSliderWidget_ValueChangedEvent valueChangedEvent;
  GFX_Point handleDownOffset;
} laSliderWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laSliderState state;	slider state
laSliderOrientation alignment;	slider alignment
int32_t min;	slider min value
int32_t max;	slider max value
int32_t value;	slider current value
uint32_t grip;	slider grip size
laSliderWidget_ValueChangedEvent valueChangedEvent;	value changed event

Description

Structure: laSliderWidget_t

A slider bar is a widget that is capable of displaying a range and a slider handle. The slider can be moved between two discreet values and can have a variable min and max range.

Remarks

None.

laSliderWidget_ValueChangedEvent Type

File

libaria_widget_slider.h

C

```
typedef void (* laSliderWidget_ValueChangedEvent)(laSliderWidget*);
```

Description

This is type laSliderWidget_ValueChangedEvent.

laTextFieldWidget Structure

Implementation of a text field widget.

File

libaria_widget_textfield.h

C

```
typedef struct laTextFieldWidget_t {
  laEditWidget editWidget;
  laString text;
  laHAlignment halign;
  uint32_t cursorPos;
  uint32_t cursorDelay;
  uint32_t cursorTime;
  laBool cursorEnable;
  laBool cursorVisible;
  laBool clearOnFirstEdit;
  laTextFieldWidget_TextChangedCallback textChangedEvent;
  GFXU_ExternalAssetReader* reader;
} laTextFieldWidget;
```

Members

Members	Description
laEditWidget editWidget;	edit widget base class

laString text;	the text to edit
laHAlignment halign;	horizontal alignment
uint32_t cursorPos;	current cursor position
uint32_t cursorDelay;	cursor blink delay
uint32_t cursorTime;	current cursor tick counter
laBool cursorEnable;	cursor enabled flag
laBool cursorVisible;	cursor visbility flag
laBool clearOnFirstEdit;	needs clear on first edit
laTextFieldWidget_TextChangedCallback textChangedEvent;	text changed event
GFXU_ExternalAssetReader* reader;	asset reader

Description

Enumeration: laTextFieldWidget_t

A text field widget is a widget that is capable of displaying a single line of editable text. This widget is capable of receiving edit events from the Aria edit event system. It can also display a blinking cursor.

Remarks

None.

laTextFieldWidget_TextChangedCallback Type

File

libaria_widget_textfield.h

C

```
typedef void (* laTextFieldWidget_TextChangedCallback)(laTextFieldWidget*);
```

Description

This is type laTextFieldWidget_TextChangedCallback.

laTouchState Structure

Manage the touch input state and track the touch coordinate

File

libaria_input.h

C

```
typedef struct laTouchState_t {
  uint32_t valid;
  int32_t x;
  int32_t y;
} laTouchState;
```

Description

Structure: laTouchState

Manage the touch input state and track the touch coordinate

Remarks

None.

laTouchTestState Enumeration

Touch test states

File

libaria_widget_touchtest.h

C

```
typedef enum laTouchTestState_t {
```

```
  LA_TOUCHTEST_STATE_UP,
  LA_TOUCHTEST_STATE_DOWN
} laTouchTestState;
```

Description

Enumeration: laTouchTestState_t

Remarks

None.

laTouchTestWidget Structure

Implementation of a touch test widget struct

File

libaria_widget_touchtest.h

C

```
typedef struct laTouchTestWidget_t {
  laWidget widget;
  laTouchTestState state;
  GFX_Point pnts[LA_TOUCHTEST_MEMORY_SIZE];
  uint32_t size;
  uint32_t start;
  uint32_t next;
  laTouchTestWidget_PointAddedEventCallback cb;
} laTouchTestWidget;
```

Members

Members	Description
laWidget widget;	widget base class
laTouchTestState state;	touch test state
GFX_Point pnts[LA_TOUCHTEST_MEMORY_SIZE];	touch point array
uint32_t size;	current number of valid touch points
uint32_t start;	first valid touch point
uint32_t next;	next available touch point entry
laTouchTestWidget_PointAddedEventCallback cb;	point added callback

Description

Structure: laTouchTestWidget_t

The touch test widget is a specialized widget that displays intersecting lines based on input events. This can help visualize touch interaction and aid determining accurate input coordinates.

Remarks

None.

laTouchTestWidget_PointAddedEventCallback Type

File

libaria_widget_touchtest.h

C

```
typedef void (* laTouchTestWidget_PointAddedEventCallback)(laTouchTestWidget*, GFX_Point*);
```

Description

This is type laTouchTestWidget_PointAddedEventCallback.

laWidget Structure

Specifies Graphics widget structure to manage all properties and events associated with the widget

File

libaria_widget.h

C

```
typedef struct laWidget_t {
  uint32_t id;
  laWidgetType type;
  laBool editable;
  laBool visible;
  laBool enabled;
  GFX_Rect rect;
  uint32_t cornerRadius;
  laMargin margin;
  laBorderType borderType;
  laBackgroundType backgroundType;
  uint32_t optimizationFlags;
  uint32_t drawCount;
  GFX_PixelBuffer* cache;
  laBool cacheInvalid;
  laBool alphaEnabled;
  uint32_t alphaAmount;
  uint32_t dirtyState;
  uint32_t drawState;
  laWidget_DrawFunction_FnPtr drawFunc;
  laScheme* scheme;
  laBool root;
  laWidget* parent;
  laArray children;
  laWidget_Destructor_FnPtr destructor;
  laWidget_Moved_FnPtr moved;
  laWidget_Resized_FnPtr resized;
  laWidget_Focus_FnPtr focusGained;
  laWidget_Focus_FnPtr focusLost;
  laWidget_Update_FnPtr update;
  laWidget_Paint_FnPtr paint;
  laWidget_TouchDownEvent_FnPtr touchDown;
  laWidget_TouchUpEvent_FnPtr touchUp;
  laWidget_TouchMovedEvent_FnPtr touchMoved;
  laWidget_LanguageChangingEvent_FnPtr languageChangeEvent;
  laWidget_InvalidateBorderAreas_FnPtr invalidateBorderAreas;
} laWidget;
```

Members

Members	Description
uint32_t id;	the id of the widget
laWidgetType type;	the type of the widget
laBool editable;	indicates if this widget implements the editable interface
laBool visible;	the widget visible flag
laBool enabled;	the widget enabled flag
GFX_Rect rect;	the bounding rectangle of the widget
uint32_t cornerRadius;	corner radius, draws round corners if > 0
laMargin margin;	the margin settings for the widget
laBorderType borderType;	the widget border type
laBackgroundType backgroundType;	the widget background type
uint32_t optimizationFlags;	optimization flags
uint32_t drawCount;	number of times this widget has been drawn for the active screen
GFX_PixelBuffer* cache;	the local framebuffer cache for the widget this can be used to avoid costly parent redraw operations at the cost of using more memory
laBool cacheInvalid;	indicates that the local cache is invalid and needs to be refilled
laBool alphaEnabled;	indicates that the global alpha blending setting is enabled for this widget
uint32_t alphaAmount;	the global alpha amount to apply to this widget (cumulative with parent widgets)
uint32_t dirtyState;	the widget's dirty state
uint32_t drawState;	the widget's draw state
laWidget_DrawFunction_FnPtr drawFunc;	the next draw function to call
laScheme* scheme;	the widget's color scheme

laBool root;	indicates if this widget is a root widget
laWidget* parent;	pointer to the widget's parent
laArray children;	pointers for the widget's children
laWidget_Destructor_FnPtr destructor;	the widget's destructor
laWidget_Moved_FnPtr moved;	moved function pointer
laWidget_Resized_FnPtr resized;	resized function pointer
laWidget_Focus_FnPtr focusGained;	focus gained function pointer
laWidget_Focus_FnPtr focusLost;	focus lost function pointer
laWidget_Update_FnPtr update;	update function pointer
laWidget_Paint_FnPtr paint;	paint function pointer
laWidget_TouchDownEvent_FnPtr touchDown;	touch down function pointer
laWidget_TouchUpEvent_FnPtr touchUp;	touch up function pointer
laWidget_TouchMovedEvent_FnPtr touchMoved;	touch moved function pointer
laWidget_LanguageChangingEvent_FnPtr languageChangeEvent;	languaged event pointer

Description

Structure: laWidget_t

Specifies Graphics widget structure to manage all properties and events associated with the widget. It also contains information about the parent and children for the widget to manage the tree structure that the library supports.

Remarks

None.

laWidget_Constructor_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Constructor_FnPtr)(laWidget*);
```

Description

This is type laWidget_Constructor_FnPtr.

laWidget_Destructor_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Destructor_FnPtr)(laWidget*);
```

Description

This is type laWidget_Destructor_FnPtr.

laWidget_DrawFunction_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_DrawFunction_FnPtr)(void*);
```

Description

This is type laWidget_DrawFunction_FnPtr.

laWidget_Focus_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Focus_FnPtr)(laWidget*);
```

Description

This is type laWidget_Focus_FnPtr.

laWidget_Moved_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Moved_FnPtr)(laWidget*);
```

Description

This is type laWidget_Moved_FnPtr.

laWidget_Paint_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Paint_FnPtr)(laWidget*);
```

Description

This is type laWidget_Paint_FnPtr.

laWidget_Resized_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_Resized_FnPtr)(laWidget*);
```

Description

This is type laWidget_Resized_FnPtr.

laWidget_TouchDownEvent_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_TouchDownEvent_FnPtr)(laWidget*, laInput_TouchDownEvent*);
```

Description

This is type laWidget_TouchDownEvent_FnPtr.

laWidget_TouchMovedEvent_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_TouchMovedEvent_FnPtr)(laWidget*, laInput_TouchMovedEvent*);
```

Description

This is type laWidget_TouchMovedEvent_FnPtr.

laWidget_TouchUpEvent_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_TouchUpEvent_FnPtr)(laWidget*, laInput_TouchUpEvent*);
```

Description

This is type laWidget_TouchUpEvent_FnPtr.

laWidget_Update_FnPtr Type

File

libaria_widget.h

C

```
typedef laWidgetUpdateState (* laWidget_Update_FnPtr)(laWidget*, uint32_t);
```

Description

This is type laWidget_Update_FnPtr.

laWidgetDirtyState Enumeration

Specifies the different dirty states the widget can be assigned

File

libaria_widget.h

C

```
typedef enum laWidgetDirtyState_t {
  LA_WIDGET_DIRTY_STATE_CLEAN,
  LA_WIDGET_DIRTY_STATE_CHILD,
  LA_WIDGET_DIRTY_STATE_DIRTY
} laWidgetDirtyState;
```

Description

Enumeration: laWidgetDirtyState_t

Specifies the different dirty states the widget can be assigned This decides whether the particular widget would be re-drawn or not. Dirty widget are re-drawn and clean are not painted over.

Remarks

None.

laWidgetDrawState Enumeration

Specifies the different draw states the widget can be assigned

File

libaria_widget.h

C

```
typedef enum laWidgetDrawState_t {
  LA_WIDGET_DRAW_STATE_READY,
  LA_WIDGET_DRAW_STATE_DONE
} laWidgetDrawState;
```

Description

Enumeration: laWidgetDrawState_t

Specifies the different draw states the widget can be assigned

Remarks

None.

laWidgetEvent Structure

Basic widget event definition

File

libaria_event.h

C

```
typedef struct laWidgetEvent_t {
  laEventID id;
  laWidget* source;
  laWidget* target;
  laBool accepted;
} laWidgetEvent;
```

Description

Structure: laWidgetEvent_t

laWidgetType Enumeration

Specifies the different widget types used in the library

File

libaria_widget.h

C

```
typedef enum laWidgetType_t {
  LA_WIDGET_WIDGET,
  LA_WIDGET_LAYER,
  LA_WIDGET_ARC,
  LA_WIDGET_BAR_GRAPH,
  LA_WIDGET_BUTTON,
  LA_WIDGET_CHECKBOX,
  LA_WIDGET_CIRCLE,
  LA_WIDGET_CIRCULAR_GAUGE,
  LA_WIDGET_CIRCULAR_SLIDER,
  LA_WIDGET_DRAWSURFACE,
  LA_WIDGET_IMAGE,
  LA_WIDGET_IMAGEPLUS,
  LA_WIDGET_IMAGESEQUENCE,
  LA_WIDGET_GRADIENT,
  LA_WIDGET_GROUPBOX,
  LA_WIDGET_KEYPAD,
  LA_WIDGET_LABEL,
  LA_WIDGET_LINE,
  LA_WIDGET_LINE_GRAPH,
  LA_WIDGET_LIST,
  LA_WIDGET_LISTWHEEL,
  LA_WIDGET_PIE_CHART,
  LA_WIDGET_PROGRESSBAR,
  LA_WIDGET_RADIAL_MENU,
```

```
  LA_WIDGET_RADIOBUTTON,
  LA_WIDGET_RECTANGLE,
  LA_WIDGET_SCROLLBAR,
  LA_WIDGET_SLIDER,
  LA_WIDGET_TEXTFIELD,
  LA_WIDGET_TOUCHTEST,
  LA_WIDGET_WINDOW
} laWidgetType;
```

Description

Enumeration: laWidgetType_t

This enumeration specifies the different widget types used in the library.

Remarks

None.

laWindowWidget Structure

Implementation of a window widget struct

File

libaria_widget_window.h

C

```
typedef struct laWindowWidget_t {
  laWidget widget;
  laString title;
  GFXU_ImageAsset* icon;
  uint32_t iconMargin;
  struct {
    GFX_Rect barRect;
  } paintData;
  GFXU_ExternalAssetReader* reader;
} laWindowWidget;
```

Members

Members	Description
laWidget widget;	base widget class
laString title;	title text
GFXU_ImageAsset* icon;	title icon
uint32_t iconMargin;	title icon margin
GFXU_ExternalAssetReader* reader;	asset reader

Description

Structure: laWindowWidget_t

A window widget is an extension of a basic panel. It adds a title bar with text and an icon.

Remarks

None.

laBackgroundType Enumeration

Specifies the different background types used for the widgets in the library

File

libaria_widget.h

C

```
typedef enum laBackgroundType_t {
  LA_WIDGET_BACKGROUND_NONE,
  LA_WIDGET_BACKGROUND_FILL,
  LA_WIDGET_BACKGROUND_CACHE,
  LA_WIDGET_BACKGROUND_LAST = LA_WIDGET_BACKGROUND_CACHE
} laBackgroundType;
```

Description

Enumeration: laBackgroundType_t

Specifies the different background types used for the widgets in the library

None - No background fill. Widget must defer to its parent to erase dirty pixels. This may cause additional overhead as clean pixels may be repainted as well.

Fill - a scheme color is used to fill the widget rectangle.

Cache - a local framebuffer cache is maintained by the widget and used to clean up dirty pixels. Will not cause a parent repaint event but will use additional memory to contain the cache.

Remarks

None.

laWidget_LanguageChangingEvent_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_LanguageChangingEvent_FnPtr)(laWidget*);
```

Description

This is type laWidget_LanguageChangingEvent_FnPtr.

laWidgetOptimizationFlags Enumeration

Specifies the different draw optimization flags for a widget

File

libaria_widget.h

C

```
typedef enum laWidgetOptimizationFlags_t {
  LA_WIDGET_OPT_LOCAL_REDRAW = 0x1,
  LA_WIDGET_OPT_DRAW_ONCE = 0x2,
  LA_WIDGET_OPT_OPAQUE = 0x4
} laWidgetOptimizationFlags;
```

Members

Members	Description
LA_WIDGET_OPT_LOCAL_REDRAW = 0x1	local redraw If a widget has no background then normally the parent would need to redraw to erase the contents of the widget. This flag indicates to the renderer to not redraw the parent event if the widget has no background
LA_WIDGET_OPT_DRAW_ONCE = 0x2	draw once Indicates that a widget should draw once per screen show event all other attempts to invalidate or paint a widget will be rejected
LA_WIDGET_OPT_OPAQUE = 0x4	opaque Indicates that a widget is fully opaque regardless of its background setting. This is often used for cases like image widgets where the image fills the entire widget space but you don't want the overhead of drawing a background behind it as well. This flag helps widgets without backgrounds to pass occlusion tests.

Description

Enumeration: laWidgetOptimizationFlags_t

Specifies the different draw optimization flags for a widget

Remarks

None.

LA_DEFAULT_SCHEME_COLOR_MODE Macro

File

libaria_common.h

C

```
#define LA_DEFAULT_SCHEME_COLOR_MODE GFX_COLOR_MODE_RGB_565
```

Description

This is macro LA_DEFAULT_SCHEME_COLOR_MODE.

LA_STRING_NULLIDX Macro

File

libaria_string.h

C

```
#define LA_STRING_NULLIDX -1
```

Description

This is macro LA_STRING_NULLIDX.

DEFAULT_BORDER_MARGIN Macro

File

libaria_widget.h

C

```
#define DEFAULT_BORDER_MARGIN 4
```

Description

This is macro DEFAULT_BORDER_MARGIN.

LA_IMAGESEQ_RESTART Macro

File

libaria_widget_imagesequence.h

C

```
#define LA_IMAGESEQ_RESTART -1
```

Description

This is macro LA_IMAGESEQ_RESTART.

LA_INPUT_PRIMARY_ID Macro

File

libaria_input.h

C

```
#define LA_INPUT_PRIMARY_ID 0
```

Description

This is macro LA_INPUT_PRIMARY_ID.

LA_MAX_TOUCH_STATES Macro

File

libaria_input.h

C

```
#define LA_MAX_TOUCH_STATES 2
```

Description

This is macro LA_MAX_TOUCH_STATES.

LA_TOUCHTEST_MEMORY_SIZE Macro

File

libaria_widget_touchtest.h

C

```
#define LA_TOUCHTEST_MEMORY_SIZE 20
```

Description

This is macro LA_TOUCHTEST_MEMORY_SIZE.

NUM_BUTTONS Macro

File

libaria_input.h

C

```
#define NUM_BUTTONS BUTTON_LAST + 1
```

Description

This is macro NUM_BUTTONS.

NUM_KEYS Macro

File

libaria_input.h

C

```
#define NUM_KEYS KEY_LAST + 1
```

Description

This is macro NUM_KEYS.

laLayer_AddDamageRect Function

Adds a damaged rectangle to the list. Damage rectangles are used in minimal redraw algorithms.

File

libaria_layer.h

C

```
LIB_EXPORT laResult laLayer_AddDamageRect(laLayer* layer, const GFX_Rect* rect, laBool noCombine);
```

Returns

laResult - the result of the operation

Parameters

Parameters	Description
laLayer* layer	the layer
const GFX_Rect* rect	the rectangle

Function

laResult laLayer_AddDamageRect(laLayer* layer, const GFX_Rect* rect)

laEventResult Enumeration

Defines what happened when processing an event

File

libaria_event.h

C

```
typedef enum laEventResult_t {
  LA_EVENT_HANDLED,
  LA_EVENT_DEFERRED,
  LA_EVENT_RESET_QUEUE
} laEventResult;
```

Members

Members	Description
LA_EVENT_HANDLED	the event was handled
LA_EVENT_DEFERRED	the event needs to wait
LA_EVENT_RESET_QUEUE	the entire event queue should be flushed and reset

Description

Enumeration: laEventResult

laLayerFrameState Enumeration

Defines the frame state of a layer. Certain actions must only be performed at the start of a new frame and other actions must wait until the end of the current frame.

File

libaria_layer.h

C

```
typedef enum laLayerFrameState_t {
  LA_LAYER_FRAME_READY,
  LA_LAYER_FRAME_PREFRAME,
  LA_LAYER_FRAME_IN_PROGRESS,
  LA_LAYER_FRAME_COMPLETE
} laLayerFrameState;
```

Description

Enumeration: laLayerFrameState

Remarks

None.

laRectArray Type

File

libaria_widget.h

C

```
typedef struct laRectArray_t laRectArray;
```

Description

This is type laRectArray.

laWidget_InvalidateBorderAreas_FnPtr Type

File

libaria_widget.h

C

```
typedef void (* laWidget_InvalidateBorderAreas_FnPtr)(laWidget*);
```

Description

This is type laWidget_InvalidateBorderAreas_FnPtr.

laContextFrameState Enumeration

Possible values for context frame state.

File

libaria_context.h

C

```
typedef enum laContextFrameState_t {
  LA_CONTEXT_FRAME_READY = 0,
  LA_CONTEXT_FRAME_PREFRAME,
  LA_CONTEXT_FRAME_PRELAYER,
  LA_CONTEXT_FRAME_DRAWING,
  LA_CONTEXT_FRAME_POSTLAYER
} laContextFrameState;
```

Description

Enumeration: laContextFrameState_t

Possible values for context frame state.

Remarks

None.

laImageWidget_DrawEventCallback Type

File

libaria_widget_image.h

C

```
typedef void (* laImageWidget_DrawEventCallback)(laImageWidget*);
```

Section

Data Types and Constants

laContextUpdateState Enumeration

Possible values for context update state.

File

libaria_context.h

C

```
typedef enum laContextUpdateState_t {
  LA_CONTEXT_UPDATE_DONE = 0,
  LA_CONTEXT_UPDATE_PENDING
} laContextUpdateState;
```

Description

Enumeration: laContextUpdateState

Possible values for context update state.

Remarks

None.

laWidgetUpdateState Enumeration

Specifies the different update states the widget can be assigned

File

libaria_widget.h

C

```
typedef enum laWidgetUpdateState_t {
  LA_WIDGET_UPDATE_STATE_DONE,
  LA_WIDGET_UPDATE_STATE_PENDING
} laWidgetUpdateState;
```

Description

Enumeration: laWidgetUpdateState_t

Specifies the different update states the widget can be assigned

Remarks

None.

Files

Files

Name	Description
libaria_common.h	This file defines the common macros and definitions used by the gfx definition and implementation headers.
libaria_context.h	Context definition for the Aria user interface library.
libaria_draw.h	Internal standard drawing help function definitions.
libaria_editwidget.h	
libaria_event.h	Defines events that are used in the UI library. Events are created and stored for later processing during a library context's update loop.
libaria_global.h	This file contains global definitions used by the Aria user interface library.
libaria_input.h	
libaria_layer.h	Aria layers map directly to layers provided by the Graphics Hardware Abstraction layer. HAL layers map directly to hardware layers provided by graphics hardware. UI layers are logical containers for widgets and provide many of the same features.
libaria_list.h	A linked list implementation for the Aria user interface library
libaria_math.h	This is file libaria_math.h.
libaria_radiobutton_group.h	
libaria_scheme.h	A scheme is a collection of colors that can be referenced by one or more widgets. Widgets may use schemes in different ways. While the color names strive to be intuitive they aren't always used in the manner in which they describe.
libaria_screen.h	A screen describes the state of a set of layers. It can be orthogonally rotated and its life-cycle can be configured.
libaria_string.h	A string library implementation for the Aria user interface library.
libaria_utils.h	General internal utilities for the library
libaria_widget.h	
libaria_widget_button.h	Defines a button widget
libaria_widget_checkbox.h	
libaria_widget_circle.h	
libaria_widget_drawsurface.h	
libaria_widget_gradient.h	
libaria_widget_groupbox.h	
libaria_widget_image.h	
libaria_widget_imagesequence.h	
libaria_widget_keypad.h	
libaria_widget_label.h	
libaria_widget_line.h	

libaria_widget_list.h	
libaria_widget_listwheel.h	
libaria_widget_progressbar.h	
libaria_widget_radiobutton.h	
libaria_widget_rectangle.h	
libaria_widget_scrollbar.h	
libaria_widget_slider.h	
libaria_widget_textfield.h	
libaria_widget_touchtest.h	
libaria_widget_window.h	Window Widget

Description

libaria_common.h

This file defines the common macros and definitions used by the gfx definition and implementation headers.

Enumerations

	Name	Description
	laBool_t	libaria bool values
	laPreemptionLevel	libaria pre-emption level values
	laRelativePosition	libaria relative position values
	laResult_t	libaria results (success and failure codes).
	laBool	libaria bool values
	laHAlignment	libaria horizontal alignment values
	laResult	libaria results (success and failure codes).
	laVAlignment	libaria vertical alignment values

Macros

	Name	Description
	LA_DEFAULT_SCHEME_COLOR_MODE	This is macro LA_DEFAULT_SCHEME_COLOR_MODE.

Structures

	Name	Description
	laMargin_t	libaria margin values
	laMargin	libaria margin values

Description

Module for Microchip Graphics Library - Aria User Interface Library

This file defines the common macros and definitions used by the gfx definition and the implementation header.

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

libaria_common.h

Company

Microchip Technology Inc.

libaria_context.h

Context definition for the Aria user interface library.

Enumerations

	Name	Description
	laContextFrameState_t	Possible values for context frame state.

	laContextUpdateState_t	Possible values for context update state.
	laContextFrameState	Possible values for context frame state.
	laContextUpdateState	Possible values for context update state.

Functions

	Name	Description
	laContext_AddScreen	Add screen to the list of screens in the current context
	laContext_Create	Creates an instance of the Aria user interface library
	laContext_Destroy	Destroys an Aria instance
	laContext_GetActive	Returns the current active context.
	laContext_GetActiveScreen	Returns the active screen of the current context
	laContext_GetActiveScreenIndex	Return the index of the active screen
	laContext_GetColorMode	Returns the color mode of the current context
	laContext_GetDefaultScheme	Returns the pointer to the default scheme of the current context
	laContext_GetEditWidget	Gets the widget that is currently recieving all widget edit events.
	laContext_GetFocusWidget	Return a pointer to the widget in focus
	laContext_GetPremptionLevel	Returns the preemption level for the screen
	laContext_GetScreenRect	Returns the display rectangle structure of the physical display
	laContext_GetStringLanguage	Returns the language index of the current context
	laContext_GetStringTable	Get a pointer to the GFXU_StringTableAsset structure that maintains the strings, associated fonts, etc
	laContext_HideActiveScreen	Hide the active screen
	laContext_IsDrawing	Indicates if any layers of the active screen are currently drawing a frame.
	laContext_IsLayerDrawing	Indicates if the layer at the given index of the active screen is currently drawing.
	laContext_RedrawAll	Forces the library to redraw the currently active screen in its entirety.
	laContext_RemoveScreen	Remove the specified screen from the list of screens in the current context
	laContext_SetActive	Make the specified context active
	laContext_SetActiveScreen	Change the active screen to the one specified by the index argument
	laContext_SetActiveScreenChangedCallback	Set the callback function pointer when the screen change event occurs
	laContext_SetEditWidget	Sets the currently active edit widget.
	laContext_SetFocusWidget	Set into focus the widget specified as the argument
	laContext_SetLanguageChangedCallback	Set the callback function pointer when the language change event occurs
	laContext_SetPreemptionLevel	Set the preemption level to the specified value
	laContext_SetStringLanguage	Set the language index of the current context
	laContext_SetStringTable	Set the StringTable pointer to the specified new StringTableAsset structure
	laContext_Update	Runs the update loop for a library instance.

Structures

	Name	Description
	laContext_t	An instance of the Aria user interface library.

Types

	Name	Description
	laContext_ActiveScreenChangedCallback_FnPtr	Callback pointer for the active screen change notification.
	laContext_LanguageChangedCallback_FnPtr	Callback pointer for when the language change event occurs.

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_context.h

Company

Microchip Technology Inc.

libaria_draw.h

Internal standard drawing help function definitions.

Functions

	Name	Description
	laDraw_1x2BevelBorder	Internal utility function to draw a 1x2 bevel border
	laDraw_2x1BevelBorder	Internal utility function to draw a 2x1 bevel border
	laDraw_2x2BevelBorder	Internal utility function to draw a 2x2 bevel border
	laDraw_LineBorder	Internal utility function to draw a basic line border

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_draw.h

Company

Microchip Technology Inc.

libaria_editwidget.h

Functions

	Name	Description
	laEditWidget_Accept	This is function laEditWidget_Accept.
	laEditWidget_Append	This is function laEditWidget_Append.
	laEditWidget_Backspace	This is function laEditWidget_Backspace.
	laEditWidget_Clear	This is function laEditWidget_Clear.
	laEditWidget_EndEdit	This is function laEditWidget_EndEdit.
	laEditWidget_Set	This is function laEditWidget_Set.
	laEditWidget_StartEdit	This is function laEditWidget_StartEdit.

Structures

	Name	Description
	laEditWidget_t	Specifies the edit widget structure to manage all properties and events associated with edit widgets
	laEditWidget	Specifies the edit widget structure to manage all properties and events associated with edit widgets

Types

	Name	Description
	laEditWidget_Accept_FnPtr	This is type laEditWidget_Accept_FnPtr.
	laEditWidget_Append_FnPtr	This is type laEditWidget_Append_FnPtr.
	laEditWidget_Backspace_FnPtr	This is type laEditWidget_Backspace_FnPtr.
	laEditWidget_Clear_FnPtr	This is type laEditWidget_Clear_FnPtr.
	laEditWidget_EndEdit_FnPtr	This is type laEditWidget_EndEdit_FnPtr.
	laEditWidget_Set_FnPtr	This is type laEditWidget_Set_FnPtr.
	laEditWidget_StartEdit_FnPtr	This is type laEditWidget_StartEdit_FnPtr.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements the routines to enable edit of library widgets.

File Name

libaria_editwidget.h

Company

Microchip Technology Inc.

libaria_event.h

Defines events that are used in the UI library. Events are created and stored for later processing during a library context's update loop.

Enumerations

	Name	Description
	laEventID_t	Defines internal event type IDs
	laEventResult_t	Defines what happened when processing an event
	laEventID	Defines internal event type IDs
	laEventResult	Defines what happened when processing an event

Functions

	Name	Description
	laEvent_AddEvent	Add the mentioned event callback to the list of events maintained by the current context
	laEvent_ClearList	Clear the event list maintained by the current context.
	laEvent_GetCount	Returns the number of events listed in the current context
	laEvent_ProcessEvents	Processes the screen change as well as touch events
	laEvent_SetFilter	Set callback pointer for current context filter event

Structures

	Name	Description
	laEvent_t	Basic UI event definition
	laEventState_t	Structure to manage the event lists, state and call back pointers
	laWidgetEvent_t	Basic widget event definition
	laEvent	Basic UI event definition
	laEventState	Structure to manage the event lists, state and call back pointers
	laWidgetEvent	Basic widget event definition

Types

	Name	Description
	laEvent_FilterEvent	Function pointer to define an event filter. Event filters allow a receiver to discard undesirable events

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_event.h

Company

Microchip Technology Inc.

libaria_global.h

This file contains global definitions used by the Aria user interface library.

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_global.h

Company

Microchip Technology Inc.

libaria_input.h

Enumerations

	Name	Description
	laGestureID_t	Placeholder for eventual gesture support.
	laKey_t	All values possible for key entry from the libaria keyboard widget
	laMouseButton_t	All values possible for mouse key entry from the libaria mouse input
	laGestureID	Placeholder for eventual gesture support.
	laKey	All values possible for key entry from the libaria keyboard widget
	laMouseButton	All values possible for mouse key entry from the libaria mouse input

Functions

	Name	Description
	laInput_GetEnabled	Returns the input enabled status of the current context
	laInput_InjectTouchDown	Register and track the touch down event and queue it for handling by associated widgets
	laInput_InjectTouchMoved	Register and track the touch moved event and queue it for handling by associated widgets
	laInput_InjectTouchUp	Register and track the touch up event and queue it for handling by associated widgets
	laInput_SetEnabled	Sets the input status of the current context with the specified input argument

Macros

	Name	Description
	LA_INPUT_PRIMARY_ID	This is macro LA_INPUT_PRIMARY_ID.
	LA_MAX_TOUCH_STATES	This is macro LA_MAX_TOUCH_STATES.
	NUM_BUTTONS	This is macro NUM_BUTTONS.
	NUM_KEYS	This is macro NUM_KEYS.

Structures

	Name	Description
	laInput_TouchDownEvent_t	Register and handle the touch press detect event
	laInput_TouchMovedEvent_t	Register and handle the touch coordinates changed event
	laInput_TouchUpEvent_t	Register and handle the touch release detect event
	laInputState_t	Maintain a history of touch states; currently libaria keeps track of the last touch state only.
	laTouchState_t	Manage the touch input state and track the touch coordinate
	laInput_TouchDownEvent	Register and handle the touch press detect event
	laInput_TouchMovedEvent	Register and handle the touch coordinates changed event
	laInput_TouchUpEvent	Register and handle the touch release detect event
	laInputState	Maintain a history of touch states; currently libaria keeps track of the last touch state only.
	laTouchState	Manage the touch input state and track the touch coordinate

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_input.h

Company

Microchip Technology Inc.

libaria_layer.h

Aria layers map directly to layers provided by the Graphics Hardware Abstraction layer. HAL layers map directly to hardware layers provided by graphics hardware. UI layers are logical containers for widgets and provide many of the same features.

Enumerations

	Name	Description
	laLayerBufferType_t	Defines the type of a layer. If the layer has an explicit address then Aria tries to set that through the HAL when the layer is being set up.
	laLayerFrameState_t	Defines the frame state of a layer. Certain actions must only be performed at the start of a new frame and other actions must wait until the end of the current frame.
	laLayerBufferType	Defines the type of a layer. If the layer has an explicit address then Aria tries to set that through the HAL when the layer is being set up.
	laLayerFrameState	Defines the frame state of a layer. Certain actions must only be performed at the start of a new frame and other actions must wait until the end of the current frame.

Functions

	Name	Description
	laLayer_AddDamageRect	Adds a damaged rectangle to the list. Damage rectangles are used in minimal redraw algorithms.
	laLayer_Delete	Destructor for the layer object
	laLayer_GetAllowInputPassThrough	Gets the layer's input passthrough setting
	laLayer_GetAlphaAmount	Get's the amount of alpha blending for a given layer
	laLayer_GetAlphaEnable	Gets the layer alpha enable flag
	laLayer_GetBufferCount	Return the buffer count for the current layer
	laLayer_GetEnabled	Returns the boolean value of the layer enabled property
	laLayer_GetInputRect	Gets the layer's input rectangle.
	laLayer_GetInputRectLocked	Gets the layer's input rect locked flag
	laLayer_GetMaskColor	Returns the mask color value for the current layer
	laLayer_GetMaskEnable	Gets the layer mask enable flag
	laLayer_GetVSync	Gets the layer's vsync flag setting
	laLayer_IsDrawing	Queries a layer to find out if it is currently drawing a frame.
	laLayer_New	Constructor for a new layer
	laLayer_SetAllowInputPassthrough	Sets the layer's input passthrough flag.
	laLayer_SetAlphaAmount	Set's the amount of alpha blending for a given layer
	laLayer_SetAlphaEnable	Sets the layer alpha enable flag to the specified value
	laLayer_SetBufferCount	Set the buffer count for the current layer to the specified value
	laLayer_SetEnabled	Sets the boolean value of the layer enabled property
	laLayer_SetInputRect	Sets the layer's input rect dimensions.
	laLayer_SetInputRectLocked	Sets the layer's input rect locked flag.
	laLayer_SetMaskColor	Set the mask color value for the current layer to the specified value
	laLayer_SetMaskEnable	Sets the layer mask enable flag to the specified value
	laLayer_SetVSync	Sets the layer's vsync flag.

Structures

	Name	Description
	laLayer_t	Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.
	laLayerBuffer_t	Structure to maintain the buffer type and track the buffer location for each layer
	laLayerBuffer	Structure to maintain the buffer type and track the buffer location for each layer

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_layer.h

Company

Microchip Technology Inc.

libaria_list.h

A linked list implementation for the Aria user interface library

Functions

	Name	Description
	laList_Assign	Assignes a new pointer to an index in the list
	laList_Clear	Removes all nodes from a given list
	laList_Copy	Creates a duplicate of an existing list
	laList_Create	Initializes a new linked list
	laList_Destroy	Removes all nodes from a given list and frees the data of each node
	laList_Find	Retrieves the index of a value from the list
	laList_Get	Retrieves a value from the list
	laList_InsertAt	Inserts an item into a list at a given index. All existing from index are shifted right one place.
	laList_PopBack	Removes the last value from the list
	laList_PopFront	Removes the first value from the list
	laList_PushBack	Pushes a new node onto the back of the list
	laList_PushFront	Pushes a new node onto the front of the list
	laList_Remove	Removes an item from the list
	laList_RemoveAt	Removes an item from the list at an index

Structures

	Name	Description
	laList_t	Linked list definition
	laListNode_t	Linked list node definition
	laListNode	Linked list node definition

Description

Module for Microchip Graphics Library - Aria User Interface Library

This is a linked list implementation that is used internally by the Aria user interface library. All of the memory operations are handled by the memory interface that is provided by the active libaria context. Applications that wish to use this implementation must ensure that the appropriate libaria context is active when calling these functions.

File Name

libaria_list.h

Company

Microchip Technology Inc.

libaria_math.h

This is file libaria_math.h.

libaria_radiobutton_group.h

Functions

	Name	Description
	laRadioButtonGroup_AddButton	Add a button widget to the button list of the selected Radio button group.
	laRadioButtonGroup_Create	This function creates a GFX_GOL_RADIOBUTTON group with the provided button list.
	laRadioButtonGroup_Destroy	This function destroys the GFX_GOL_RADIOBUTTON group
	laRadioButtonGroup_RemoveButton	Remove a button widget to the button list of the selected Radio button group.
	laRadioButtonGroup_SelectButton	Select the button widget specified from the button list for the Radio button group.

Structures

	Name	Description
	laRadioButtonGroup_t	Defines the structure used for the Radio Button group.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements functions to control a radio button group.

File Name

libaria_radiobutton_group.h

Company

Microchip Technology Inc.

libaria_scheme.h

A scheme is a collection of colors that can be referenced by one or more widgets. Widgets may use schemes in different ways. While the color names strive to be intuitive they aren't always used in the manner in which they describe.

Functions

	Name	Description
	laScheme_Initialize	Initialize the scheme to the default values as per the specified color mode.

Structures

	Name	Description
	laScheme_t	This structure specifies the style scheme components of an object.
	laScheme	This structure specifies the style scheme components of an object.

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_scheme.h

Company

Microchip Technology Inc.

libaria_screen.h

A screen describes the state of a set of layers. It can be orthogonally rotated and its life-cycle can be configured.

Enumerations

	Name	Description
	laScreenOrientation_t	Possible values for screen orientation.
	laScreenOrientation	Possible values for screen orientation.

Functions

	Name	Description
	laScreen_Delete	Frees all memory for all layers and widgets for this screen
	laScreen_GetHideEventCallback	Returns the hide call back event function pointer for the specified screen
	laScreen_GetLayerIndex	Returns the index of the layer for the screen specified.
	laScreen_GetLayerSwapSync	Returns the layer swap sync setting for the specified screen
	laScreen_GetMirrored	Returns the mirror setting for the specified screen
	laScreen_GetOrientation	Returns the orientation object associated with the specified screen
	laScreen_GetShowEventCallback	Returns the show call back event function pointer for the specified screen
	laScreen_Hide	Hide the currently active screen This function has been deprecated in favor of laContext_SetActiveScreen
	laScreen_New	Create a new screen, initialize it to the values specified.
	laScreen_SetHideEventCallback	Set the hide call back event function pointer for the specified screen
	laScreen_SetLayer	Assigns the provided layer pointer to the screen at the given index This function has been deprecated in favor of laContext_SetActiveScreen
	laScreen_SetLayerSwapSync	Sets the layer swap sync setting for the specified screen

	laScreen_SetMirrored	Sets the mirror setting for the specified screen
	laScreen_SetOrientation	Sets the orientation object to the specified screen
	laScreen_SetShowEventCallback	Set the show call back event function pointer for the specified screen
	laScreen_Show	Make the specified screen active and show it on the display

Structures

	Name	Description
	laScreen_t	The structure to maintain the screen related variables and event handling
	laScreen	The structure to maintain the screen related variables and event handling

Types

	Name	Description
	laScreen_CreateCallback_FnPtr	Callback pointer for a new screen create event notification. This is called when the library attempts to create a screen.
	laScreen_ShowHideCallback_FnPtr	Callback pointer for the active screen show or hide event change notification.

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_screen.h

Company

Microchip Technology Inc.

libaria_string.h

A string library implementation for the Aria user interface library.

Functions

	Name	Description
	laString_Allocate	Attempts to resize the local data buffer for a string.
	laString_Append	Appends a string onto the end of another string
	laString_Capacity	Returns the capacity of a string
	laString_CharAt	Extracts the code point for the character in a string at a given index.
	laString_Clear	Sets a string's length to zero and its string table reference to NULL. Does not free any associated data and preserves capacity.
	laString_Compare	Compares two string objects
	laString_CompareBuffer	Compares a string object and a GFXU_CHAR* buffer
	laString_Copy	Copies the values from one string into another
	laString_CreateFromBuffer	Creates a string object from a GFXU_CHAR buffer and a font asset pointer
	laString_CreateFromCharBuffer	Creates a string object from a const char* buffer and a font asset pointer. This method provides compatibility with standard c-style strings. Input string will be converted from 8-bit with to 32-bit width.
	laString_CreateFromID	Creates a string object that simply references a string in the string table.
	laString_Delete	Deletes all memory associated with a string object
	laString_Destroy	Destroys a string object. This frees the strings internal data buffer, if it exists, sets its string table reference to null, and clears all supporting attributes.
	laString_Draw	Wrapper around GFX Utility string draw function for Aria user interface library. Internal use only.
	laString_DrawClipped	Wrapper around GFX Utility string draw function for Aria user interface library. Draws only a clipped area of a string. Internal use only.
	laString_DrawSubStringClipped	Wrapper around GFX Utility string draw function for Aria user interface library. Draws the substring between the start and end offset, and draws only the section of the string within the clipping rectangle. Internal use only.
	laString_ExtractFromTable	Extracts a read-only string from the string table into a modifiable string object. This relies on the active context to indicate which string table to reference as well as which language entry to extract.
	laString_GetCharIndexAtPoint	Given an offset in pixels returns the corresponding character index.

	laString_GetCharOffset	Returns the offset of a given character index in pixels.
	laString_GetCharWidth	Given a character index, gets the width of that character. Only accurate if the string has a font associated with it and that font contains all the characters in the string in question.
	laString_GetHeight	Returns the height of a string by referencing its associated font asset data.
	laString_GetLineRect	Calculates the rectangle for a line in a string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The line ends when a line feed or end of string is reached.
	laString_GetMultiLineRect	Calculates the rectangle for a given multi-line string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup. The height the sum of the heights of the bounding rectangles for each line and the width is the widest among the bounding rectangles.
	laString_GetRect	Calculates the rectangle for a given string object. References the associated font for the height but must perform a summation for each character in the string by doing a font meta-data lookup.
	laString_Initialize	Initializes a string struct to default
	laString_Insert	Inserts a string into another string at a given index
	laString_IsEmpty	Returns a boolean indicating if the provided string contains data or has a link to the string table.
	laString_Length	Calculates the length of a string in characters
	laString_New	Allocates a memory for a new string
	laString_ReduceLength	Reduces the length of a string. This simply slides the null terminator to the left and does not affect the string's capacity value.
	laString_Remove	Removes a number of characters from a string at a given index
	laString_Set	Attempts to set the local data buffer of a string to an input buffer
	laString_SetCapacity	Attempts to adjust the capacity of a string
	laString_ToCharBuffer	Extracts the data buffer from a string and copies it into the provided buffer argument.

Macros

	Name	Description
	LA_STRING_NULLIDX	This is macro LA_STRING_NULLIDX.

Structures

	Name	Description
	laString_t	String definition
	laString	String definition

Types

	Name	Description
	laContext	An instance of the Aria user interface library.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This is a string library implementation that is used internally by the Aria user interface library. All of the memory operations are handled by the memory interface that is provided by the active libaria context. Applications that wish to use this implementation must ensure that the appropriate libaria context is active when calling these functions.

This implementation relies on the GFXU_CHAR definition for characters provided by the GFX Utils library. This chararcter definition is 32 bits in size and allows libaria to support international character code points and Unicode encoding standards.

File Name

libaria_string.h

Company

Microchip Technology Inc.

libaria_utils.h

General internal utilities for the library

Functions

	Name	Description
	laUtils_ArrangeRectangle	Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the image rectangle.
	laUtils_ArrangeRectangleRelative	Calculates the position of a rectangle within the given bounds and in accordance with the given parameters. A use case for this is when an image and a text rectangle must be arranged in a button box. This version of the algorithm will calculate the location of the text rectangle.
	laUtils_ChildIntersectsParent	Performs an intersection test between a parent widget and a child widget
	laUtils_ClipRectToParent	Clips a rectangle to the parent of a widget
	laUtils_GetLayer	Finds the root parent of a widget, which should be a layer
	laUtils_GetNextHighestWidget	Gets the next highest Z order widget in the tree from 'wgt'
	laUtils_ListOcclusionCullTest	Performs an occlusion test on a list of widgets an a rectangular area. This attempts to find only the topmost widgets for the given area. If a widget is completely occluded then it is removed from the list. Any widgets that remain in the list should be redrawn by the rasterizer.
	laUtils_OcclusionCullTest	Performs an occlusion test for a widget in the tree. A widget is occluded if it is completely covered by one or more widgets. This is useful for culling widgets before the rasterizing phase.
	laUtils_Pick	Finds the top-most visible widget in the tree at the given coordinates.
	laUtils_PickFromLayer	Finds the top-most visible widget in a layer at the given coordinates.
	laUtils_PickRect	Finds all of the visible widgets in the given rectangular area.
	laUtils_PointScreenToLocalSpace	Converts a point from layer space into the local space of a widget
	laUtils_PointToLayerSpace	Converts a point from widget space into layer space
	laUtils_RectFromLayerSpace	Converts a rectangle from layer space to widget local space
	laUtils_RectFromParentSpace	Converts a rectangle from widget parent space to widget local space
	laUtils_RectToLayerSpace	Converts a rectangle from widget local space to layer space
	laUtils_RectToParentSpace	Converts a rectangle from widget local space to widget parent space. Widget must be a child of a layer for this to function.
	laUtils_RectToScreenSpace	Converts a rectangle from widget local space to screen space
	laUtils_ScreenToMirroredSpace	Takes a point in screen space and returns a transformed version in mirrored space.
	laUtils_ScreenToOrientedSpace	Takes a point in screen space and returns a transformed version in oriented space.
	laUtils_WidgetIsOccluded	This is function laUtils_WidgetIsOccluded.
	laUtils_WidgetLayerRect	Returns the bounding rectangle of a widget in layer space
	laUtils_WidgetLocalRect	Returns the bounding rectangle of a widget in local space

Types

	Name	Description
	GFX_Point	A two dimensional Cartesian point.
	GFX_Rect	A rectangle definition.
	laLayer	Primary definition of a layer. Builds on base functions of a standard widget. Should never have a direct parent.
	laList	Linked list definition

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_utils.h

Company

Microchip Technology Inc.

libaria_widget.h

Enumerations

	Name	Description
	laBackgroundType_t	Specifies the different background types used for the widgets in the library

	laBorderType_t	Specifies the different border types used for the widgets in the library
	laWidgetDirtyState_t	Specifies the different dirty states the widget can be assigned
	laWidgetDrawState_t	Specifies the different draw states the widget can be assigned
	laWidgetOptimizationFlags_t	Specifies the different draw optimization flags for a widget
	laWidgetType_t	Specifies the different widget types used in the library
	laWidgetUpdateState_t	Specifies the different update states the widget can be assigned
	laBackgroundType	Specifies the different background types used for the widgets in the library
	laBorderType	Specifies the different border types used for the widgets in the library
	laWidgetDirtyState	Specifies the different dirty states the widget can be assigned
	laWidgetDrawState	Specifies the different draw states the widget can be assigned
	laWidgetOptimizationFlags	Specifies the different draw optimization flags for a widget
	laWidgetType	Specifies the different widget types used in the library
	laWidgetUpdateState	Specifies the different update states the widget can be assigned

Functions

	Name	Description
	laWidget_AddChild	Adds the child to the parent widget specified in the argument
	laWidget_Delete	Delete the widget object specified
	laWidget_DeleteAllDescendants	Deletes all of the descendants of the given widget.
	laWidget_GetAlphaAmount	Return the widget's global alpha amount
	laWidget_GetAlphaEnable	Return the alpha enable property of the widget
	laWidget_GetBackgroundType	Return the property value 'background type' associated with the widget object
	laWidget_GetBorderType	Return the border type associated with the widget object
	laWidget_GetChildAtIndex	Fetches the child at the specified index from the children list of the specified parent widget
	laWidget_GetChildCount	Returns the size of the children list of the specified parent widget
	laWidget_GetCumulativeAlphaAmount	Calculates the cumulative alpha amount for a hierarchy of widgets
	laWidget_GetCumulativeAlphaEnable	Determines if this or any ancestor widget has alpha enabled
	laWidget_GetEnabled	Returns the boolean value of the widget enabled property
	laWidget_GetHeight	Returns the widget rectangles height
	laWidget_GetIndexOfChild	Fetches the index of the child from the children list of the specified parent widget
	laWidget_GetMargin	Returns the margin value associated with the widget in the laMargin pointer
	laWidget_GetOptimizationFlags	Returns the optimization flags for the widget
	laWidget_GetScheme	Returns the scheme associated with the specified widget
	laWidget_GetVisible	Returns the boolean value of the widget visible property
	laWidget_GetWidth	Returns the widget rectangles width
	laWidget_GetX	Returns the widget rectangles upper left corner x-coordinate
	laWidget_GetY	Returns the widget rectangles upper left corner y-coordinate
	laWidget_HasFocus	Checks if the widget specified has focus in the current context
	laWidget_Invalidate	Invalidates the specified widget.
	laWidget_isOpaque	Returns true if the widget is considered opaque.
	laWidget_New	Create a new widget.
	laWidget_OverrideTouchDownEvent	Replace the TouchDownEvent callback for the widget with the new function pointer specified
	laWidget_OverrideTouchMovedEvent	Replace the TouchMovedEvent callback for the widget with the new function pointer specified
	laWidget_OverrideTouchUpEvent	Replace the TouchUpEvent callback for the widget with the new function pointer specified
	laWidget_RectToLayerSpace	Transforms a widget rectangle from local space to its root layer space.
	laWidget_RectToParentSpace	Returns the rectangle containing the parent of the widget specified
	laWidget_RectToScreenSpace	Transforms a widget rectangle from local space to screen space coordinates.
	laWidget_RemoveChild	Removes the child from the parent widget specified in the argument
	laWidget_Resize	Changes the widget size by the new defined width and height increments.
	laWidget_SetAlphaAmount	Set the widget's global alpha amount to the specified alpha amount
	laWidget_SetAlphaEnable	Set the alpha enable property of the widget with the boolean value specified
	laWidget_SetBackgroundType	Set the property value 'background type' associated with the widget object
	laWidget_SetBorderType	Set the border type associated with the widget object

	laWidget_SetEnabled	Sets the boolean value of the widget enabled property
	laWidget_SetFocus	Set the widget into focus for the current context.
	laWidget_SetHeight	Sets the widget's height value
	laWidget_SetMargins	Set the margin value for left, right, top and bottom margins associated with the widget
	laWidget_SetOptimizationFlags	Sets the optimizations for a widget
	laWidget_SetParent	Sets the parent of the child widget to that specified in the argument list
	laWidget_SetPosition	Changes the widget position to the new defined x and y coordinates.
	laWidget_SetScheme	Sets the scheme variable for the specified widget
	laWidget_SetSize	Changes the widget size to the new defined width and height dimensions.
	laWidget_SetVisible	Sets the boolean value of the widget visible property
	laWidget_SetWidth	Sets the widget's width value
	laWidget_SetX	Sets the widget's x coordinate position
	laWidget_SetY	Sets the widget's y coordinate position
	laWidget_Translate	Changes the widget position by moving the widget by the defined x and y increments.

Macros

	Name	Description
	DEFAULT_BORDER_MARGIN	This is macro DEFAULT_BORDER_MARGIN.

Structures

	Name	Description
	laWidget_t	Specifies Graphics widget structure to manage all properties and events associated with the widget
	laWidget	Specifies Graphics widget structure to manage all properties and events associated with the widget

Types

	Name	Description
	laRectArray	This is type laRectArray.
	laWidget_Constructor_FnPtr	This is type laWidget_Constructor_FnPtr.
	laWidget_Destructor_FnPtr	This is type laWidget_Destructor_FnPtr.
	laWidget_DrawFunction_FnPtr	This is type laWidget_DrawFunction_FnPtr.
	laWidget_Focus_FnPtr	This is type laWidget_Focus_FnPtr.
	laWidget_InvalidateBorderAreas_FnPtr	This is type laWidget_InvalidateBorderAreas_FnPtr.
	laWidget_LanguageChangingEvent_FnPtr	This is type laWidget_LanguageChangingEvent_FnPtr.
	laWidget_Moved_FnPtr	This is type laWidget_Moved_FnPtr.
	laWidget_Paint_FnPtr	This is type laWidget_Paint_FnPtr.
	laWidget_Resized_FnPtr	This is type laWidget_Resized_FnPtr.
	laWidget_TouchDownEvent_FnPtr	This is type laWidget_TouchDownEvent_FnPtr.
	laWidget_TouchMovedEvent_FnPtr	This is type laWidget_TouchMovedEvent_FnPtr.
	laWidget_TouchUpEvent_FnPtr	This is type laWidget_TouchUpEvent_FnPtr.
	laWidget_Update_FnPtr	This is type laWidget_Update_FnPtr.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements top level widget control functions.

File Name

libaria_widget.h

Company

Microchip Technology Inc.

libaria_widget_button.h

Defines a button widget

Enumerations

	Name	Description
	laButtonState_t	Controls the button pressed state
	laButtonState	Controls the button pressed state

Functions

	Name	Description
	laButtonWidget_GetHAlignment	Gets the horizontal alignment setting for a button
	laButtonWidget_GetImageMargin	Gets the distance between the icon and the text
	laButtonWidget_GetImagePosition	Gets the position of the button icon
	laButtonWidget_GetPressed	Gets the pressed state of a button
	laButtonWidget_GetPressedEventCallback	Gets the callback associated with the button pressed event
	laButtonWidget_GetPressedImage	Gets the pressed image asset pointer for a button
	laButtonWidget_GetPressedOffset	Gets the offset of the button internals when pressed
	laButtonWidget_GetReleasedEventCallback	Gets the callback for the button released event
	laButtonWidget_GetReleasedImage	Gets the currently used released icon
	laButtonWidget_GetText	Gets the text for a button. If the button's string has local data then a duplicate of the string will be allocated. The caller is responsible for managing the memory for the duplicated string. If the button string is a string table reference then only the reference ID is copied.
	laButtonWidget_GetToggleable	Gets the value of this button's toggle flag
	laButtonWidget_GetVAlignment	Gets the vertical alignment setting for a button
	laButtonWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laButtonWidget_SetHAlignment	Sets the horizontal alignment value for a button
	laButtonWidget_SetImageMargin	Sets the distance between the icon and text
	laButtonWidget_SetImagePosition	Sets the position of the button icon
	laButtonWidget_SetPressed	Sets the pressed state for a button.
	laButtonWidget_SetPressedEventCallback	Sets the pressed event callback for the button
	laButtonWidget_SetPressedImage	Sets the image to be used as a pressed icon
	laButtonWidget_SetPressedOffset	Sets the offset of the button internals when pressed
	laButtonWidget_SetReleasedEventCallback	Sets the callback for the button released event
	laButtonWidget_SetReleasedImage	Sets the image to be used as the released icon
	laButtonWidget_SetText	Sets the text for a button. If the input string has local data then the data will be copied into the button's local string, causing a memory allocation. If the input string is a string table reference then only the reference will be copied. The input string can be safely modified and the button string will not be affected.
	laButtonWidget_SetToggleable	Enables the toggle mode for a button. When pressed, toggle buttons will stay down until pressed again.
	laButtonWidget_SetVAlignment	Sets the vertical alignment for a button

Structures

	Name	Description
	laButtonWidget_t	Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.

Types

	Name	Description
	laButtonWidget_PressedEvent	This is type laButtonWidget_PressedEvent.
	laButtonWidget_ReleasedEvent	This is type laButtonWidget_ReleasedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

File Name

libaria_widget_button.h

Company

Microchip Technology Inc.

libaria_widget_checkbox.h

Functions

	Name	Description
	laCheckBoxWidget_GetChecked	Gets the checked state of the check box
	laCheckBoxWidget_GetCheckedEventCallback	Gets the checked event callback
	laCheckBoxWidget_GetCheckedImage	Gets the checked image of the check box
	laCheckBoxWidget_GetHAlignment	Gets the horizontal alignment of the check box
	laCheckBoxWidget_GetImageMargin	Gets the distance between the image and the text
	laCheckBoxWidget_GetImagePosition	Gets the image position of the check box
	laCheckBoxWidget_GetText	Gets a copy of the checkbox text. If the text has local data the data will be duplicated. The caller is responsible for managing the memory as appropriate.
	laCheckBoxWidget_GetUncheckedEventCallback	Gets the unchecked event callback
	laCheckBoxWidget_GetUncheckedImage	Gets the unchecked image of the check box
	laCheckBoxWidget_GetVAlignment	Gets the vertical alignment of the check box
	laCheckBoxWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laCheckBoxWidget_SetChecked	Sets the checked state of the check box
	laCheckBoxWidget_SetCheckedEventCallback	Sets the checked event callback
	laCheckBoxWidget_SetCheckedImage	Sets the checked image of the check box
	laCheckBoxWidget_SetHAlignment	Sets the horizontal alignment of the check box.
	laCheckBoxWidget_SetImageMargin	Sets the distance between the image and the text
	laCheckBoxWidget_SetImagePosition	Sets the image position of the check box
	laCheckBoxWidget_SetText	Sets the checkbox text to the input string. If the string has local data the data will be duplicated and copied to the checkboxes internal string.
	laCheckBoxWidget_SetUncheckedEventCallback	Sets the unchecked event callback
	laCheckBoxWidget_SetUncheckedImage	Sets the unchecked image of the check box
	laCheckBoxWidget_SetVAlignment	Sets the vertical alignment of the check box

Structures

	Name	Description
	laCheckBoxWidget_t	Implementation of a checkbox widget.
	laCheckBoxWidget	Implementation of a checkbox widget.

Types

	Name	Description
	laCheckBoxWidget_CheckedEvent	This is type laCheckBoxWidget_CheckedEvent.
	laCheckBoxWidget_UncheckedEvent	This is type laCheckBoxWidget_UncheckedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements button widget functions.

File Name

libaria_widget_button.h

Company

Microchip Technology Inc.

libaria_widget_circle.h

Functions

	Name	Description
	laCircleWidget_GetOrigin	Gets the origin coordiates of a circle widget
	laCircleWidget_GetRadius	Gets the radius of a circle widget
	laCircleWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laCircleWidget_SetOrigin	Sets the origin coordiates of a circle widget
	laCircleWidget_SetRadius	Sets the radius of a circle widget

Structures

	Name	Description
	laCircleWidget_t	Implementation of a circle widget.
	laCircleWidget	Implementation of a circle widget.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements circle drawing widget functions.

File Name

libaria_widget_circle.h

Company

Microchip Technology Inc.

libaria_widget_drawsurface.h

Functions

	Name	Description
	laDrawSurfaceWidget_GetDrawCallback	Returns the pointer to the currently set draw callback.
	laDrawSurfaceWidget_New	Allocates memory for a new DrawSurface widget.
	laDrawSurfaceWidget_SetDrawCallback	Sets the draw callback pointer for the draw surface widget.

Structures

	Name	Description
	laDrawSurfaceWidget_t	Implementation of a Drawsurface widget.
	laDrawSurfaceWidget	Implementation of a Drawsurface widget.

Types

	Name	Description
	laDrawSurfaceWidget_DrawCallback	This is type laDrawSurfaceWidget_DrawCallback.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements surface container drawing functions.

File Name

libaria_widget_drawsurface.h

Company

Microchip Technology Inc.

libaria_widget_gradient.h

Enumerations

	Name	Description
	laGradientWidgetDirection_t	Implementation of a gradient widget.
	laGradientWidgetDirection	Implementation of a gradient widget.

Functions

	Name	Description
	laGradientWidget_GetDirection	Gets the gradient direction value for this widget.
	laGradientWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laGradientWidget_SetDirection	Sets the gradient direction value for this widget.

Structures

	Name	Description
	laGradientWidget_t	Gradient widget struct definition.
	laGradientWidget	Gradient widget struct definition.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements gradient drawing widget functions.

File Name

libaria_widget_gradient.h

Company

Microchip Technology Inc.

libaria_widget_groupbox.h

Functions

	Name	Description
	laGroupBoxWidget_GetAlignment	Gets the horizontal alignmnet for the group box title text
	laGroupBoxWidget_GetText	Gets the text value for the group box.
	laGroupBoxWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laGroupBoxWidget_SetAlignment	Sets the alignment for the group box title text
	laGroupBoxWidget_SetText	Sets the text value for the group box.

Structures

	Name	Description
	laGroupBoxWidget_t	Group box struct definition.
	laGroupBoxWidget	Group box struct definition.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements group box widget functions.

File Name

libaria_widget_groupbox.h

Company

Microchip Technology Inc.

libaria_widget_image.h

Functions

	Name	Description
	laImageWidget_GetHAlignment	Gets the image horizontal alignment value.
	laImageWidget_GetImage	Gets the image asset pointer for the widget.
	laImageWidget_GetVAlignment	Gets the image vertical alignment value.
	laImageWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laImageWidget_SetCallBackEnd	This is function laImageWidget_SetCallBackEnd.
	laImageWidget_SetCallBackStart	This is function laImageWidget_SetCallBackStart.
	laImageWidget_SetHAlignment	Sets the image horizontal alignment value.
	laImageWidget_SetImage	Sets the image asset pointer for the widget.
	laImageWidget_SetVAlignment	Sets the image vertical alignment value.

Structures

	Name	Description
	laImageWidget_t	Image widget struct definition
	laImageWidget	Image widget struct definition

Types

	Name	Description
	laImageWidget_DrawEventCallback	

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements image widget functions.

File Name

libaria_widget_image.h

Company

Microchip Technology Inc.

libaria_widget_imagesequence.h

Functions

	Name	Description
	laImageSequenceWidget_GetImage	Gets the image asset pointer for an entry.
	laImageSequenceWidget_GetImageChangedEventCallback	Gets the image changed event callback pointer.
	laImageSequenceWidget_GetImageCount	Gets the number of image entries for this widget.
	laImageSequenceWidget_GetImageDelay	Gets the image delay for an entry.
	laImageSequenceWidget_GetImageHAlignment	Gets the horizontal alignment for an image entry
	laImageSequenceWidget_GetImageVAlignment	Sets the vertical alignment for an image entry
	laImageSequenceWidget_GetRepeat	Indicates if the widget will repeat through the image entries.
	laImageSequenceWidget_IsPlaying	Indicates if the widget is currently cycling through the image entries.
	laImageSequenceWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laImageSequenceWidget_Play	Starts the widget automatically cycling through the image entries.
	laImageSequenceWidget_Rewind	Resets the current image sequence display index to zero.
	laImageSequenceWidget_SetImage	Sets the image asset pointer for an entry.
	laImageSequenceWidget_SetImageChangedEventCallback	Sets the image changed event callback pointer. This callback is called whenever the active display index is changed.
	laImageSequenceWidget_SetImageCount	Sets the number of image entries for this widget. An image entry that is null will show nothing.

	laImageSequenceWidget_SetImageDelay	Sets the image delay for an entry.
	laImageSequenceWidget_SetImageHAlignment	Sets the horizontal alignment for an image entry.
	laImageSequenceWidget_SetImageVAlignment	Sets the vertical alignment value for an image entry
	laImageSequenceWidget_SetRepeat	Sets the repeat flag for the widget
	laImageSequenceWidget_ShowImage	Sets the active display index to the indicated value.
	laImageSequenceWidget_ShowNextImage	Sets the active display index to the next index value.
	laImageSequenceWidget_ShowPreviousImage	Sets the active display index to the previous index value.
	laImageSequenceWidget_Stop	Stops the widget from automatically cycling through the image entries.

Macros

	Name	Description
	LA_IMAGESEQ_RESTART	This is macro LA_IMAGESEQ_RESTART.

Structures

	Name	Description
	laImageSequenceEntry_t	Image sequence entry definition
	laImageSequenceWidget_t	Image sequence widget struct definition
	laImageSequenceEntry	Image sequence entry definition
	laImageSequenceWidget	Image sequence widget struct definition

Types

	Name	Description
	laImageSequenceImageChangedEvent_FnPtr	This is type laImageSequenceImageChangedEvent_FnPtr.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements image sequence (slide show) widget drawing functions.

File Name

libaria_widget_imagesequence.h

Company

Microchip Technology Inc.

libaria_widget_keypad.h

Enumerations

	Name	Description
	laKeyPadCellAction_t	Defines an assigned action to a key pad cell
	laKeyPadCellAction	Defines an assigned action to a key pad cell

Functions

	Name	Description
	laKeyPadWidget_GetKeyAction	Gets the key pad cell action for a cell at row/column
	laKeyPadWidget_GetKeyClickEventCallback	Gets the current key click event callback pointer
	laKeyPadWidget_GetKeyDrawBackground	Gets the background type for a key pad cell at row/column
	laKeyPadWidget_GetKeyEnabled	Gets the enabled flag for a cell at a given row/column
	laKeyPadWidget_GetKeyImageMargin	Gets the key pad cell image margin value
	laKeyPadWidget_GetKeyImagePosition	Gets the image position for a key pad cell
	laKeyPadWidget_GetKeyPressedImage	Gets the pressed icon image asset pointer for the display image for a key pad cell
	laKeyPadWidget_GetKeyReleasedImage	Gets the released icon image asset pointer for the display image for a key pad cell
	laKeyPadWidget_GetKeyText	Returns a copy of the display text for a given cell at row/column
	laKeyPadWidget_GetKeyValue	Gets the edit text value for a given key pad cell.
	laKeyPadWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laKeyPadWidget_SetKeyAction	Sets the cell action type for a key pad cell at row/column

	laKeyPadWidget_SetKeyBackgroundType	Sets the background type for a key pad cell at row/column
	laKeyPadWidget_SetKeyClickEventCallback	Sets the current key click event callback pointer
	laKeyPadWidget_SetKeyEnabled	Sets the enabled flag for a cell at the given row/column
	laKeyPadWidget_SetKeyImageMargin	Sets the key pad cell image margin value for a given cell at row/column
	laKeyPadWidget_SetKeyImagePosition	
	laKeyPadWidget_SetKeyPressedImage	Sets the pressed icon image asset pointer for a key pad cell
	laKeyPadWidget_SetKeyReleasedImage	Sets the released icon image asset pointer for a key pad cell
	laKeyPadWidget_SetKeyText	Sets the display text for a given cell at row/column
	laKeyPadWidget_SetKeyValue	Sets the edit value for a given key pad cell.

Structures

	Name	Description
	laKeyPadCell_t	Defines a key pad cell struct
	laKeyPadWidget_t	Defines a key pad widget struct
	laKeyPadCell	Defines a key pad cell struct
	laKeyPadWidget	Defines a key pad widget struct

Types

	Name	Description
	laButtonWidget	Implementation of a button widget. A button is an interactive element that simulates a typical button with a pressed an released state.
	laKeyPadWidget_KeyClickEvent	This is type laKeyPadWidget_KeyClickEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements keypad widget functions.

File Name

libaria_widget_keypad.h

Company

Microchip Technology Inc.

libaria_widget_label.h

Functions

	Name	Description
	laLabelWidget_GetHAlignment	Gets the text horizontal alignment value.
	laLabelWidget_GetText	Gets the text value for the label.
	laLabelWidget_GetVAlignment	Gets the current vertical text alignment
	laLabelWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laLabelWidget_SetHAlignment	Sets the text horizontal alignment value
	laLabelWidget_SetText	Sets the text value for the label.
	laLabelWidget_SetVAlignment	Sets the vertical text alignment value

Structures

	Name	Description
	laLabelWidget_t	Implementation of a label widget struct
	laLabelWidget	Implementation of a label widget struct

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements label (text) widget functions.

File Name

libaria_widget_label.h

Company

Microchip Technology Inc.

libaria_widget_line.h

Functions

	Name	Description
	laLineWidget_GetEndPoint	Gets the coordinates for the second point of the line.
	laLineWidget_GetStartPoint	Gets the coordinates for the first point of the line.
	laLineWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laLineWidget_SetEndPoint	Sets the cooridnate for the second point of the line
	laLineWidget_SetStartPoint	Sets the cooridnate for the first point of the line

Structures

	Name	Description
	laLineWidget_t	Defines the implementation of a line widget struct
	laLineWidget	Defines the implementation of a line widget struct

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements line draw widget functions.

File Name

libaria_widget_line.h

Company

Microchip Technology Inc.

libaria_widget_list.h

Enumerations

	Name	Description
	laListWidget_SelectionMode_t	Defines the list selection modes
	laListWidget_SelectionMode	Defines the list selection modes

Functions

	Name	Description
	laListWidget_AppendItem	Appends a new item entry to the list. The initial value of the item will be empty.
	laListWidget_DeselectAll	Attempts to set all item states as not selected.
	laListWidget_GetAlignment	Gets the horizontal alignment for the list widget
	laListWidget_GetAllowEmptySelection	Returns true of the list allows an empty selection set
	laListWidget_GetFirstSelectedItem	Returns the lowest selected item index.
	laListWidget_GetIconMargin	Gets the icon margin value for the list widget
	laListWidget_GetIconPosition	Gets the icon position for the list
	laListWidget_GetItemCount	Gets the number of items currently contained in the list
	laListWidget_GetItemIcon	Gets the pointer to the image asset for the icon for the item at the given index.
	laListWidget_GetItemSelected	Returns true if the item at the given index is currently selected.
	laListWidget_GetItemText	Gets the text value for an item in the list.
	laListWidget_GetLastSelectedItem	Returns the highest selected item index.
	laListWidget_GetSelectedItemChangedEventCallback	Gets the callback for the item selected changed event
	laListWidget_GetSelectionCount	Returns the number of selected items in the list.
	laListWidget_GetSelectionMode	Gets the selection mode for the list

	laListWidget_InsertItem	Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.
	laListWidget_New	Allocates memory for and initializes a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laListWidget_RemoveAllItems	Attempts to remove all items from the list.
	laListWidget_RemoveItem	Attempts to remove an item from the list.
	laListWidget_SelectAll	Attempts to set all item states to selected.
	laListWidget_SetAlignment	Sets the horizontal alignment mode for the list widget.
	laListWidget_SetAllowEmptySelection	Configures the list to allow an empty selection set.
	laListWidget_SetIconMargin	Sets the icon margin value for the list widget.
	laListWidget_SetIconPosition	Sets the icon position for the list widget
	laListWidget_SetItemIcon	Sets the icon pointer for a given index.
	laListWidget_SetItemSelected	Attempts to set the item at idx as selected.
	laListWidget_SetItemText	Sets the text value for an item in the list.
	laListWidget_SetItemVisible	
	laListWidget_SetSelectedItemChangedEventCallback	Sets the callback for the item selected changed event
	laListWidget_SetSelectionMode	Set the list selection mode
	laListWidget_ToggleItemSelected	Attempts to toggle the selected state of the item at idx.

Structures

	Name	Description
	laListItem_t	Defines a list item struct
	laListWidget_t	Defines the implementation of a list widget
	laListItem	Defines a list item struct
	laListWidget	Defines the implementation of a list widget

Types

	Name	Description
	laListWidget_ItemSelectedChangedEvent	This is type laListWidget_ItemSelectedChangedEvent.
	laListWidget_SelectedItemChangedEvent	This is type laListWidget_SelectedItemChangedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements list box widget functions.

File Name

libaria_widget_list.h

Company

Microchip Technology Inc.

libaria_widget_listwheel.h

Functions

	Name	Description
	laListWheelWidget_AppendItem	Appends a new item entry to the list. The initial value of the item will be empty.
	laListWheelWidget_GetAlignment	Gets the horizontal alignment for the list widget
	laListWheelWidget_GetFlickInitSpeed	Returns the flick init speed for the wheel.
	laListWheelWidget_GetIconMargin	Gets the icon margin value for the list wheel widget
	laListWheelWidget_GetIconPosition	Sets the icon position for the list wheel widget.
	laListWheelWidget_GetIndicatorArea	Returns the spacing for the selected item indicator bars.
	laListWheelWidget_GetItemCount	Gets the number of items currently contained in the list
	laListWheelWidget_GetItemIcon	Gets the pointer to the image asset for the icon for the item at the given index.
	laListWheelWidget_GetItemText	Gets the text value for an item in the list.

	laListWheelWidget_GetMaxMomentum	Returns the maximum momentum value for the wheel.
	laListWheelWidget_GetMomentumFalloffRate	Returns the momentum falloff rate for the wheel.
	laListWheelWidget_GetRotationUpdateRate	Returns the wheel rotation update rate.
	laListWheelWidget_GetSelectedItem	Returns the index of the currently selected item.
	laListWheelWidget_GetSelectedItemChangedEventCallback	Gets the callback for the item selected changed event
	laListWheelWidget_GetShaded	Returns true if the list is using gradient shading to illustrate depth
	laListWheelWidget_GetShowIndicators	Returns true if the list is displaying its selected item indicators
	laListWheelWidget_GetVisibleItemCount	Returns the list's visible item count
	laListWheelWidget_InsertItem	Attempts to insert a new item at the desired index. Existing items at idx or greater will be shuffled one index to the right.
	laListWheelWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laListWheelWidget_RemoveAllItems	Attempts to remove all items from the list.
	laListWheelWidget_RemoveItem	Attempts to remove an item from the list.
	laListWheelWidget_SelectNextItem	Attempts to move the selected item index to the next item in the list.
	laListWheelWidget_SelectPreviousItem	Attempts to move the selected item index to the previous item in the list.
	laListWheelWidget_SetAlignment	Sets the horizontal alignment mode for the list widget.
	laListWheelWidget_SetFlickInitSpeed	Configures the flick init speed for the list wheel
	laListWheelWidget_SetIconMargin	Sets the icon margin value for the list widget.
	laListWheelWidget_SetIconPosition	Sets the icon position for the list wheel widget
	laListWheelWidget_SetIndicatorArea	Configures the display area for the list selection indicator bars
	laListWheelWidget_SetItemIcon	Sets the icon pointer for a given index.
	laListWheelWidget_SetItemText	Sets the text value for an item in the list.
	laListWheelWidget_SetMaxMomentum	Configures the maximum momentum value for the wheel
	laListWheelWidget_SetMomentumFalloffRate	Configures the momentum falloff rate for the wheel
	laListWheelWidget_SetRotationUpdateRate	Configures the rotation update rate for a wheel
	laListWheelWidget_SetSelectedItem	Attempts to set the selectedi tem index
	laListWheelWidget_SetSelectedItemChangedEventCallback	
	laListWheelWidget_SetShaded	Configures the list to use gradient or flat background shading
	laListWheelWidget_SetShowIndicators	Configures the list to display the selected item indicator bars
	laListWheelWidget_SetVisibleItemCount	Sets the number of visible items in the list. Must be greater than or equal to three and must be an odd number.

Structures

	Name	Description
	laListWheelItem_t	Implementation of a list wheel widget item struct
	laListWheelWidget_t	Implementation of a list wheel widget struct
	laListWheelItem	Implementation of a list wheel widget item struct
	laListWheelWidget	Implementation of a list wheel widget struct

Types

	Name	Description
	laListWheelWidget_SelectedItemChangedEvent	This is type laListWheelWidget_SelectedItemChangedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements list wheel (drawing-style list box) widget functions.

File Name

libaria_widget_listwheel.h

Company

Microchip Technology Inc.

libaria_widget_progressbar.h

Enumerations

	Name	Description
	laProgressBarDirection_t	Defines the valid values for the progress bar widget fill directions.
	laProgressBarDirection	Defines the valid values for the progress bar widget fill directions.

Functions

	Name	Description
	laProgressBarWidget_GetDirection	Gets the fill direction value for a progress bar widget
	laProgressBarWidget_GetValue	Gets the percentage value for a progress bar.
	laProgressBarWidget_GetValueChangedEventCallback	Gets the currently set value changed event callback.
	laProgressBarWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laProgressBarWidget_SetDirection	Sets the fill direction for a progress bar widget
	laProgressBarWidget_SetValue	Sets the percentage value for a progress bar. Valid values are 0 - 100.
	laProgressBarWidget_SetValueChangedCallback	Sets the desired value changed event callback pointer

Structures

	Name	Description
	laProgressBarWidget_t	Implementation of a progressbar widget struct
	laProgressBarWidget	Implementation of a progressbar widget struct

Types

	Name	Description
	laProgressBar	This is type laProgressBar.
	laProgressBar_ValueChangedEventCallback	This is type laProgressBar_ValueChangedEventCallback.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements progress bar widget functions.

File Name

libaria_widget_progressbar.h

Company

Microchip Technology Inc.

libaria_widget_radiobutton.h

Functions

	Name	Description
	laRadioButtonWidget_GetDeselectedEventCallback	Gets the current radio button deselected event callback
	laRadioButtonWidget_GetGroup	Returns the pointer to the currently set radio button group.
	laRadioButtonWidget_GetHAlignment	Gets the horizontal alignment setting for a button
	laRadioButtonWidget_GetImageMargin	Gets the distance between the icon and the text
	laRadioButtonWidget_GetImagePosition	Gets the current image position setting for the radio button
	laRadioButtonWidget_GetSelected	Returns true if this radio button is currently selected
	laRadioButtonWidget_GetSelectedEventCallback	Gets the current radio button selected event callback
	laRadioButtonWidget_GetSelectedImage	Gets the selected image asset pointer for a button
	laRadioButtonWidget_GetText	Gets the text value for the button.
	laRadioButtonWidget_GetUnselectedImage	Gets the image asset pointer currently used as the unselected icon
	laRadioButtonWidget_GetVAlignment	Sets the vertical alignment for a button
	laRadioButtonWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.

	laRadioButtonWidget_SetDeselectedEventCallback	Sets the deselected callback pointer
	laRadioButtonWidget_SetHAlignment	Sets the horizontal alignment value for a button
	laRadioButtonWidget_SetImageMargin	Sets the distance between the icon and text
	laRadioButtonWidget_SetImagePosition	Sets the image relative position setting for the radio button
	laRadioButtonWidget_SetSelected	Sets this button as selected.
	laRadioButtonWidget_SetSelectedEventCallback	Sets the radio button selected event callback
	laRadioButtonWidget_SetSelectedImage	Sets the image to be used as a selected icon
	laRadioButtonWidget_SetText	Sets the text value for the button.
	laRadioButtonWidget_SetUnselectedImage	Sets the asset pointer for the radio button's unselected image icon
	laRadioButtonWidget_SetVAlignment	Sets the vertical alignment for a button

Structures

	Name	Description
	laRadioButtonWidget_t	Implementation of a radio button widget struct
	laRadioButtonWidget	Implementation of a radio button widget struct

Types

	Name	Description
	laRadioButtonGroup	Defines the structure used for the Radio Button group.
	laRadioButtonWidget_DeselectedEvent	This is type laRadioButtonWidget_DeselectedEvent.
	laRadioButtonWidget_SelectedEvent	This is type laRadioButtonWidget_SelectedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements radio button widget functions.

File Name

libaria_widget_radiobutton.h

Company

Microchip Technology Inc.

libaria_widget_rectangle.h

Functions

	Name	Description
	laRectangleWidget_GetThickness	Gets the rectangle border thickness setting
	laRectangleWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laRectangleWidget_SetThickness	Sets the rectangle border thickness setting

Structures

	Name	Description
	laRectangleWidget_t	Implementation of a rectangle widget struct
	laRectangleWidget	Implementation of a rectangle widget struct

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements rectangle drawing widget functions.

File Name

libaria_widget_rectangle.h

Company

Microchip Technology Inc.

libaria_widget_scrollbar.h

Enumerations

	Name	Description
	laScrollBarOrientation_t	Defines the scroll bar direction values
	laScrollBarState_t	Defines the various scroll bar state values
	laScrollBarOrientation	Defines the scroll bar direction values
	laScrollBarState	Defines the various scroll bar state values

Functions

	Name	Description
	laScrollBarWidget_GetExtentValue	Gets the current scroll bar extent value
	laScrollBarWidget_GetMaxinumValue	Gets the maximum scroll value
	laScrollBarWidget_GetOrientation	Gets the orientation value for the scroll bar
	laScrollBarWidget_GetScrollPercentage	Gets the current scroll value as a percentage
	laScrollBarWidget_GetScrollValue	Gets the current scroll value
	laScrollBarWidget_GetStepSize	Gets the current discreet step size
	laScrollBarWidget_GetValueChangedEventCallback	Gets the current value changed callback function pointer
	laScrollBarWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laScrollBarWidget_SetExtentValue	Sets the scroll bar extent value
	laScrollBarWidget_SetMaximumValue	Sets the maximum scroll value
	laScrollBarWidget_SetOrientation	Sets the orientation value of the scroll bar
	laScrollBarWidget_SetScrollPercentage	Sets the current scroll value using a percentage. Percentage should be a value from 0 - 100
	laScrollBarWidget_SetScrollValue	Sets the current scroll value
	laScrollBarWidget_SetStepSize	Sets the current step size
	laScrollBarWidget_SetValueChangedEventCallback	Sets the value changed event callback pointer
	laScrollBarWidget_StepBackward	Moves the scroll value back by the current step size
	laScrollBarWidget_StepForward	Moves the scroll value forward by the current step size

Structures

	Name	Description
	laScrollBarWidget_t	Implementation of a scroll bar widget.
	laScrollBarWidget	Implementation of a scroll bar widget.

Types

	Name	Description
	laScrollBarWidget_ValueChangedEvent	This is type laScrollBarWidget_ValueChangedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements scroll bar widget functions.

File Name

libaria_widget_scrollbar.h

Company

Microchip Technology Inc.

libaria_widget_slider.h

Enumerations

	Name	Description
	laSliderOrientation_t	Slider orientations

	laSliderState_t	Describes various slider states
	laSliderOrientation	Slider orientations
	laSliderState	Describes various slider states

Functions

	Name	Description
	laSliderWidget_GetGripSize	Gets the current grip size of the slider
	laSliderWidget_GetMaxinumValue	Gets the maximum value for the slider
	laSliderWidget_GetMininumValue	Gets the minimum value for the slider
	laSliderWidget_GetOrientation	Gets the orientation value for the slider
	laSliderWidget_GetSliderPercentage	Gets the slider value as a percentage
	laSliderWidget_GetSliderValue	Gets the current slider value
	laSliderWidget_GetValueChangedEventCallback	Gets the current value changed event callback pointer
	laSliderWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laSliderWidget_SetGripSize	Sets the grip size of the slider
	laSliderWidget_SetMaximumValue	Sets the maximum value for the slider
	laSliderWidget_SetMinimumValue	Sets the minimum value for the slider
	laSliderWidget_SetOrientation	
	laSliderWidget_SetSliderPercentage	Sets the slider value using a percentage. Value must be from 0 - 100.
	laSliderWidget_SetSliderValue	Sets the current slider value
	laSliderWidget_SetValueChangedEventCallback	Sets the value changed event callback pointer
	laSliderWidget_Step	Moves the slider by a given amount

Structures

	Name	Description
	laSliderWidget_t	Implementation of a slider widget struct
	laSliderWidget	Implementation of a slider widget struct

Types

	Name	Description
	laSliderWidget_ValueChangedEvent	This is type laSliderWidget_ValueChangedEvent.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements slider control widget functions.

File Name

libaria_widget_slider.h

Company

Microchip Technology Inc.

libaria_widget_textfield.h

Functions

	Name	Description
	laTextFieldWidget_GetAlignment	Gets the text horizontal alignment value.
	laTextFieldWidget_GetCursorDelay	Gets the current cursor delay.
	laTextFieldWidget_GetCursorEnabled	Gets the cursor enabled value
	laTextFieldWidget_GetCursorPosition	Gets the current edit cursor position
	laTextFieldWidget_GetText	Gets the text value for the box.
	laTextFieldWidget_GetTextChangedEventCallback	Gets the current text changed event callback pointer
	laTextFieldWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laTextFieldWidget_SetAlignment	Sets the text horizontal alignment value

	laTextFieldWidget_SetClearOnFirstEdit	Sets the flag to indicate that the text field will be cleared on first edit.
	laTextFieldWidget_SetCursorDelay	Sets the cursor delay value
	laTextFieldWidget_SetCursorEnabled	Sets the cursor enabled value flag
	laTextFieldWidget_SetCursorPosition	Sets the position of the cursor
	laTextFieldWidget_SetText	Sets the text value for the box.
	laTextFieldWidget_SetTextChangedEventCallback	Sets the text changed event callback pointer

Structures

	Name	Description
	laTextFieldWidget_t	Implementation of a text field widget.
	laTextFieldWidget	Implementation of a text field widget.

Types

	Name	Description
	laTextFieldWidget_TextChangedCallback	This is type laTextFieldWidget_TextChangedCallback.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements text field widget functions.

File Name

libaria_widget_textfield.h

Company

Microchip Technology Inc.

libaria_widget_touchtest.h

Enumerations

	Name	Description
	laTouchTestState_t	Touch test states
	laTouchTestState	Touch test states

Functions

	Name	Description
	laTouchTest_AddPoint	Adds a point to the touch test widget. The point will then be displayed.
	laTouchTest_ClearPoints	Clears all of the existing touch points
	laTouchTestWidget_GetPointAddedEventCallback	Gets the current point added event callback
	laTouchTestWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laTouchTestWidget_SetPointAddedEventCallback	Sets the point added event callback

Macros

	Name	Description
	LA_TOUCHTEST_MEMORY_SIZE	This is macro LA_TOUCHTEST_MEMORY_SIZE.

Structures

	Name	Description
	laTouchTestWidget_t	Implementation of a touch test widget struct
	laTouchTestWidget	Implementation of a touch test widget struct

Types

	Name	Description
	laTouchTestWidget_PointAddedEventCallback	This is type laTouchTestWidget_PointAddedEventCallback.

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements graphical touch test (box) widget functions.

File Name

libaria_widget_touchtest.h

Company

Microchip Technology Inc.

libaria_widget_window.h

Window Widget

Functions

	Name	Description
	laWindowWidget_GetIcon	Gets the currently used window icon
	laWindowWidget_GetIconMargin	Gets the current image icon margin
	laWindowWidget_GetIconRect	This is function laWindowWidget_GetIconRect.
	laWindowWidget_GetTextRect	This is function laWindowWidget_GetTextRect.
	laWindowWidget_GetTitle	Gets the title text for this window.
	laWindowWidget_GetTitleBarRect	internal use only
	laWindowWidget_New	Allocates memory for a new widget of this type. The application is responsible for the managment of this memory until the widget is added to a widget tree.
	laWindowWidget_SetIcon	Sets the image to be used as a window icon
	laWindowWidget_SetIconMargin	Sets the image icon margin
	laWindowWidget_SetTitle	Sets the title text for the window.

Structures

	Name	Description
	laWindowWidget_t	Implementation of a window widget struct
	laWindowWidget	Implementation of a window widget struct

Description

Module for Microchip Graphics Library - Aria User Interface Library

This module implements window container widget functions.

File Name

libaria_widget_window.h

Company

Microchip Technology Inc.

Hardware Abstraction Layer (HAL)

This topic describes the Hardware Abstraction Layer (HAL) of the MPLAB Harmony Graphics Composer (MHGC) Suite, which is a component of MHGC Suite.

Introduction

This section introduces the Hardware Abstraction Layer (HAL) of the MPLAB Harmony Graphics Composer (MHGC) Suite.

Description

The HAL serves to abstract the details of the hardware away from the application and protect the graphics state from mismanagement. This layer is designed to be similar to industry-standard graphics APIs, such as OpenGL from SGI, and DirectX from Microsoft. Applications that use graphics should only communicate with this layer at a minimum, and should not attempt to communicate with display drivers directly.

Before looking at the operation and structure of the HAL, the following definitions of the different keywords and concepts explained within the HAL are provided.

Hardware Abstraction Layer Definitions:

- *Alpha blending*: An operation that combines two colors into a single color, based on one or more percentage values
- *Blit*: Writing an area of pixel data to a buffer

- *Buffer swap*: Cycling through a buffer chain, therefore changing the read and write buffer pointers
- *Cache coherent*: Data that must always be current in memory, such as data that is accessed by a peripheral, which should use coherent memory
- *Clipping*: Comparing a point to a rectangle, or a rectangle to a rectangle, to assess whether the point is contained inside the rectangle or conforming the area of one rectangle to fit inside another
- *Color masking*: An operation that compares a color value with a color mask value. If the values are equal, the color is ignored and not rendered to the write buffer.
- *Color mode*: Defines how pixel data is stored in memory. Some color modes consume less memory than others.
- *Context*: An instance of the hardware abstraction layer. Combines a display, a driver, and possibly a graphics accelerator.
- *Display*: A display device capable of rendering color data
- *Draw Target*: An application-defined area of memory to be used as the target for draw operations. This is often different than the active frame buffer and can be used for off-screen rendering operations.
- *Driver*: A software program that communicates directly with, and manages, hardware
- *Double buffer*: A display configuration in which multiple frame buffers are chained together to avoid screen tearing artifacts
- *Frame buffer*: An area of memory that contains pixel data. Pixel data is one of several color modes with each mode consuming various quantities of memory.
- *Heap*: A pool of memory that can be dynamically allocated
- *HSync*: A refresh state of a display device when the device is being refreshed outside the horizontal viewing area
- *Layer*: A rectangular area of space that contains one or more frame buffers. Can directly correspond to a hardware-managed layer.
- *Pixel*: A single color value stored in a predefined mode. Usually 8 to 32 bits in size.
- *Pixel buffer*: A struct that describes a rectangle of pixel data. May or may not actually contain valid pixel data.
- *Point*: A two dimensional Cartesian coordinate consisting of a horizontal “x” value and a vertical “y” value
- *Raster operation*: Any operation or algorithm that writes pixel data to the write buffer
- *Read buffer*: A buffer that is currently being used to feed display data to display hardware
- *Size*: A two dimensional measurement of magnitude consisting of a “width” and a “height” value
- *VSync*: A refresh state of a display device when the device is being refreshed outside the vertical viewing area
- *Write buffer*: A buffer that is designated as the receiver of raster operations

HAL Features

What does the Hardware Abstraction Layer do?

The HAL serves four main purposes:

- Configure abstract graphics and display hardware
- Managing frame buffer memory
- Manage draw state
- Draw shapes

Graphics and Display Hardware Configuration

The HAL serves as an intermediary between the higher level stack layers and the hardware drivers. Drivers are expected to conform to the HAL specification and applications interface with drivers through a simple set of APIs. The main purpose of this is to allow the framework to use various hardware drivers without ever having to make changes to the application. Each driver will interface with the HAL according to its specific needs and capabilities.

Frame Buffer Memory Management

Memory management is handled by the HAL for all drivers, libraries, and applications. This may include; buffer creation, buffer resizing, freeing buffer memory, buffer swapping, etc. The application simply requests buffer functions through the HAL APIs. Drivers may restrict how buffers are managed based on specific graphics controller needs and capabilities.

Draw State Management

The HAL maintains a state that indicates how raster operations should be performed.

Shape Drawing

The HAL provides APIs for basic pixel, line, circle, and rectangle drawing. These are rendered according to the draw state.

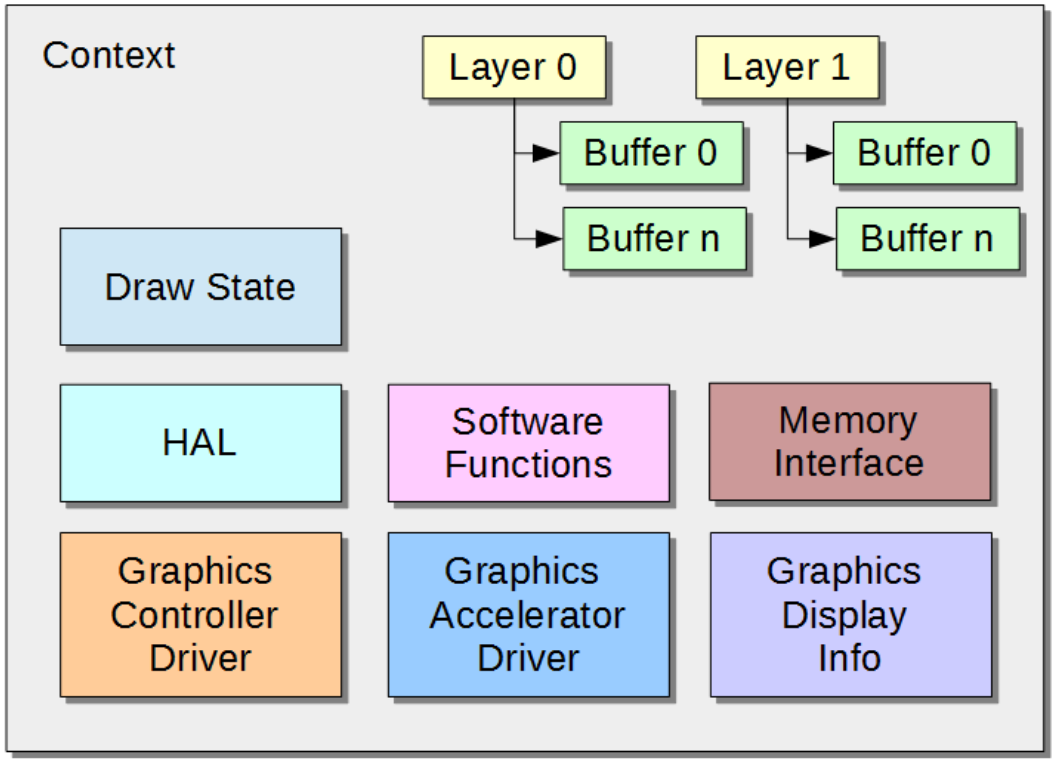
HAL Context

What is a context?

The HAL is designed to be multi-instance, which means it supports multiple drivers and displays concurrently. This is accomplished by using a *context* or *layer* state. A context is essentially the combination of a single display descriptor, and a single graphics driver. Graphics processors may also be part of a context. The HAL allows only a single context to be *active* or in use at any one time, but the application may switch contexts at any time in order to act on another state.

Note: Graphics drivers must also be able to support multi-instancing in order for the application to use multiple contexts.

The following block diagram shows a high-level representation of what the context contains. A description of each block follows the diagram.



Graphics Display Info

The application uses a “Display” definition to obtain knowledge about the displays that are available through the HAL. This definition contains information such as:

- *Name*: A short identifier for the display
- *Color Modes*: The color modes the display supports
- *Size*: A width and height
- *Timing settings*: Values for the front porches, back porches, pulse widths, etc.

Display definitions are generated through the use of HConfig and Freemarker templates. These definitions are created during the code generation phase of MHC and are expected to exist at run-time.

Graphics Controller Driver

The application uses a “Driver” definition to obtain information from the HAL about the available drivers in the system.

The driver definition consists of the following information:

- *Name*: A short identifier for the controller
- *Color Modes*: The color modes the controller supports
- *Layer Count*: The number of hardware layers the controller supports

Driver integration with the HAL will be covered in a subsequent section.

Graphics Accelerator Driver

A Graphics Processing Unit (GPU) may be present in the system. If one exists, the context must reroute GPU supported raster operations to the graphics accelerator driver for handling.

Layer

A HAL layer is a representation of a hardware based display layer most likely provided by a graphics controller. Applications are capable of using one or more of these layers up to the max value indicated by a hardware driver. Layers have a width, height, and a position in absolute space on the display.

Layers may have one or more frame buffers associated with them. Layers with two buffers are often called *double buffered*. Multiple buffers of a layer are connected to form a buffer chain, and are cycled through as needed via pointer swapping. Layers have, at all times, one buffer considered to be the *read buffer* and one buffer considered to be the *write buffer*. In a single buffer layer, the read and write buffers are the same. When drawing single buffer layers, rendering artifacts such as screen tearing may be observed. This is because a single buffer may be written to, and read from at the same time. Double buffering alleviates screen tearing as all raster operations are performed on the hidden *write buffer* and the buffers are only swapped once the *write buffer* has been fully crafted. Further, by acting during display blanking periods, the driver can swap the read and write buffer pointers during periods when the display is not actively drawing. This should completely eliminate screen tearing.

A context has one active layer at all times and all operations are performed on the active layer.

Frame Buffer

An extension of a pixel buffer, frame buffers are used by layers to track frame buffer states. Frame buffers contain a pixel buffer, but also contain the following:

- *Pixel Buffer State*: An indication of the origin of the data for the pixel buffer. This can indicate that the buffer contains no pixel data, that the pixel data was allocated from the heap, or that the buffer and associated pixel data is owned and managed by the graphics driver. The latter state is used to prevent the application from freeing buffers managed by the graphic driver.
- *Coherent*: An indication that the buffer should be allocated from cache coherent memory when it is dynamically allocated

Memory Interface

By default, the HAL uses standard library memory management functions, such as malloc, free, calloc, etc. However, in the presence of memory peripherals, the application may want the Graphics Stack to utilize a custom memory manager instead. This is accomplished by providing a memory interface definition.

This definition simply provides alternate function pointers for standard memory allocation functions.

Draw State

The context's draw state is simply a list of hints that the context feeds into raster operation functions such as a line draw. The state indicates what the draw color is, if alpha blending is enabled, if the final raster point should be adjusted for orientation or mirroring, if there is a masking or transparency color enabled, etc.

HAL

One of the most important functions of the context is to provide hardware abstraction. By default, all raster operations are handled in software, or in the Software Functions module. However, if a GPU exists, any supported raster operation requests must be rerouted to the GPU driver for handling. In other cases, the driver may need to restrict context options or handle an operation in a manner that is different from the default implementation. Therefore, the driver may change the function routing in the context's HAL state as it sees fit. However, if the driver implements non-default functionality, it must ensure that overall functionality of the context is not compromised.

Software Functions

The HAL contains a series of default implementation functions for most operations. These are represented by the Software Functions module.

Color Support

The HAL is able to create and manage a context using one of several color formats.

- GS8: 8-bit gray scale
- RGB_332: 8-bit, 256 colors
- RGB_565: 16-bit, 65536 colors
- RGB_5551: 15-bit color, 1-bit alpha, 32767 colors
- RGB_888: 24-bit color, 16 million colors
- RGBA_8888: 24-bit color, 8-bit alpha, 16 million colors
- ARGB_8888: 24-bit color 8-bit alpha, 16 million colors

All buffers that are created by the context will use this color mode. This can affect the sizes of the frame buffers that will be created.

HAL State Management

The HAL is primarily interacted with through the GFX_Get and GFX_Set functions. These variable argument functions always take as the first argument an operation ID. Then, follow a variable number of supporting arguments to either set or get data. For example:

```
GFX_Set(GFXF_DRAW_COLOR, 0xFFFF);
```

This code would set the current draw color for the active context to white, assuming a 16-bit color space. The first argument is one of the values listed in the GFX_FLAG enum and the second is the argument expected by that operation.

To get the current draw color the code would appear as follows:

```
GFX_Get(GFXF_DRAW_COLOR, &color);
```

These get and set functions can return these status values:

- GFX_FAILURE: An error occurred during this operation
- GFX_SUCCESS: The operation was successful
- GFX_UNSUPPORTED: The operation is not supported by the context

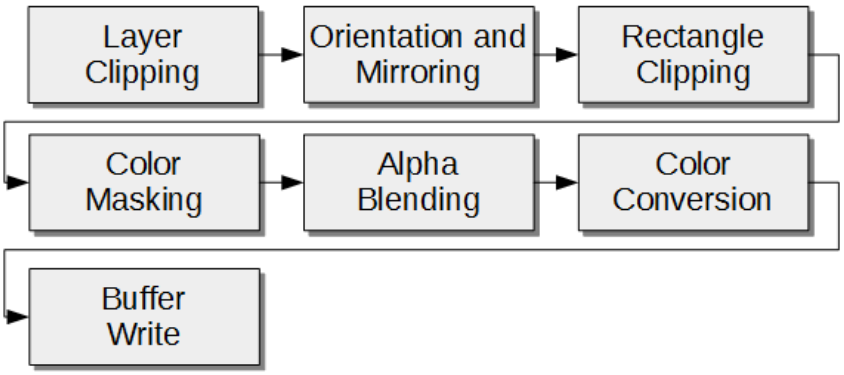
Notes:

1. All features may not be supported by all drivers. Software fall backs and default implementations may be provided for some features when hardware support is not available.
2. Most flags have a get and set mode. A few are get only and a few are set only. For detailed interface information, refer to `gfx_common.h`.

Pixel Transformation Pipeline

Overview of the Pixel Transformation Pipeline.

The HAL uses a multi-stage pixel rendering pipeline to apply the various effects that may have been enabled by the application. The stages are shown in the following figure:



Stage Description

- *Layer Clipping*: The pixel is clipped to the destination layer. It is rejected if it falls outside the layer. Writing outside of the layer can cause memory out-of-bounds exceptions.
- *Orientation and Mirroring*: The pixel's destination point is rotated and mirrored according to the draw state
- *Rectangle Clipping*: The pixel is containment evaluated with the context's clipping rectangle and rejected if it is out of bounds
- *Color Masking*: The pixel is compared to the context's color mask value. If the color matches the value it is rejected
- *Alpha Blending*: The source pixel is blended with the destination pixel. Both the color's alpha channel and the global alpha blending value is taken into consideration. A color without an alpha channel is upscaled to 32-bits and its alpha channel is set to max.
- *Color Conversion*: The color is converted to the color mode of the destination buffer. This only applies to blits.
- *Buffer Write*: The result color is written to the frame buffer at the potentially transformed point

Note:

1. These stages can be disabled in the GFX Options in the MHC option tree. Disabling them can increase speed but can cause the program to become unstable or draw incorrectly.
2. This flow is meant to show how the stages might interact but the exact order of execution is dependent on the state of the HAL and the operation being performed.

Using The Library

To access the HAL, simply include the header file `gfx.h` in your application. This is assuming that the appropriate flags have been checked in the configuration.

The HAL APIs typically fall into one of several groups:

- *Initialization*: Interfaces in the MHC configuration that are responsible for setting up the state of the HAL
- *Context Management*: Interfaces that create or destroy a graphics context
- *Context Maintenance*: Interfaces that allow the context to perform tasks such as HAL or driver state updates
- *Draw State Management*: This consists of two generic interfaces that allow the application to manage the state of a context. This is accomplished by indicating the get/set operation from a predefined list of option IDs, and sending the appropriate arguments into the variable argument functions
- *Blitting and Shape Drawing*: These interfaces perform raster operations on the active frame buffer according to the current draw state of the HAL

The following sample code displays how to initialize the HAL, create a context, create some layers and buffers, and draw a rectangle.

Note: The following code example is not performing any return value checking.

```
// context variable
GFX_Handle* context;

// initialize the HAL layer
GFX_Initialize();

// create a context.  the zeros indicate the display and driver to
// use.  the third argument would be for a custom memory interface
context = GFX_Open(0, 0, NULL);

// make sure the context is active
```

```
GFX_ContextActiveSet(context);

// set the context color mode to RGB_565
GFX_Set(GFXF_COLOR_MODE, GFX_COLOR_MODE_RGB_565);

// make sure the zeroth layer is active, enabled and visible
GFX_Set(GFXF_LAYER_ACTIVE, 0);
GFX_Set(GFXF_LAYER_ENABLED, GFX_TRUE);
GFX_Set(GFXF_LAYER_VISIBLE, GFX_TRUE);

// typically the bottom layer is going to fill the entire
// display area but for demonstration purposes change
// the position and size of the layer
GFX_Set(GFXF_LAYER_POSITION, 100, 100); // x = 100, y = 100
GFX_Set(GFXF_LAYER_SIZE, 320, 200); // width = 320, height = 200

// set the layer to two buffers and set to use coherent memory
GFX_Set(GFXF_LAYER_BUFFER_COUNT, 2);
GFX_Set(GFXF_LAYER_BUFFER_COHERENT, 0, GFX_TRUE);
GFX_Set(GFXF_LAYER_BUFFER_COHERENT, 1, GFX_TRUE);

// allocate the buffers
GFX_Set(GFXF_LAYER_BUFFER_ALLOCATE, 0);
GFX_Set(GFXF_LAYER_BUFFER_ALLOCATE, 1);

// set the draw mode and color
GFX_Set(GFXF_DRAW_MODE, GFX_DRAW_FILL);
GFX_Set(GFXF_DRAW_COLOR, 0xFFFF);

// indicate intent to draw, if this returns GFX_FAILURE then
// draw operations will fail
GFX_Begin();

// fill the entire layer with white
GFX_DrawRect(0, 0, 320, 200); // x, y, width, height

GFX_Set(GFXF_DRAW_COLOR,
        GFX_ColorValue(GFX_COLOR_MODE_RGB_565, GFX_COLOR_MAGENTA));

// draw a smaller magenta rectangle
GFX_DrawRect(10, 10, 100, 100);

// finish drawing
GFX_End();

// swap the buffers
GFX_Set(GFXF_LAYER_SWAP, GFX_TRUE);
```

Library Interface

a) Functions

	Name	Description
	GFX_AbsoluteValue	Calculates the absolute value of a signed integer.
	GFX_ActiveContext	Gets the current set active HAL context.
	GFX_Clampf	Clamps a float between a min and max
	GFX_Clampi	Clamps an integer between a min and max
	GFX_ColorChannelAlpha	Used for getting the alpha color channel of a given color value.
	GFX_ColorChannelGreen	Used for getting the green color channel of a given color value.
	GFX_ColorChannelRed	Used for getting the red color channel of a given color value.
	GFX_ColorConvert	Converts a color value from one mode to another
	GFX_ColorLerp	Linear interpolation between two colors
	GFX_ColorModeInfoGet	
	GFX_ColorValue	Used for getting a color value by name.
	GFX_ContextActiveSet	Sets the active context

	GFX_LayerReadBuffer	Gets the pointer to the layer's current read pixel buffer.
	GFX_LayerRotate	Swaps the width and height dimensions of a layer. Can be used for run-time display orientation
	GFX_LayerSwap	Performs a swap operation on the given layer. This advances the pointers of layer's buffer chain. The current write buffer becomes the new read buffer and a new buffer is chosen as the new write buffer. Has no effect in single buffer environments.
	GFX_LayerWriteBuffer	Gets the pointer to the layer's current write pixel buffer.
	GFX_Lerp	Performs a linear interpolation of an integer based on a percentage between two signed points.
	GFX_Maxf	Returns the larger of two floats.
	GFX_Maxi	Returns the larger of two integers.
	GFX_Minf	Returns the smaller of two floats.
	GFX_Mini	Returns the smaller of two integers.
	GFX_Percent	Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed. The result is the decimal percentage multiplied by 100.
	GFX_PercentOf	Calculates the percentage of a number. Returns a whole number with no decimal component.
	GFX_PercentWholeRounded	Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed. The difference between this and GFX_Percent is that the decimal portion of the whole number is rounded off.
	GFX_PixelBufferAreaFill	Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer.
	GFX_PixelBufferAreaFill_Unsafe	Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer. Like GFX_PixelBufferAreaFill but performs no bounds checking.
	GFX_PixelBufferAreaGet	Extracts a rectanglar section of pixels from a pixel buffer.
	GFX_PixelBufferAreaGet_Unsafe	Extracts a rectanglar section of pixels from a pixel buffer. Like GFX_PixelBufferAreaGet but performs no clipping between the rectangles of the extract area and the source buffer.
	GFX_PixelBufferAreaSet	Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation.
	GFX_PixelBufferAreaSet_Unsafe	Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation. Like GFX_PixelBufferAreaSet but performs no bounds checking.
	GFX_PixelBufferClipRect	Clips a rectangle against a pixel buffer. The result is guaranteed to fit inside the buffer's area.
	GFX_PixelBufferConvert	Duplicates a pixel buffer and converts the copy to another color mode.
	GFX_PixelBufferCopy	Creates a copy of the input buffer. If the input buffer's pixel data is not null then the data will be duplicated for the result buffer.
	GFX_PixelBufferCreate	Initializes a pixel buffer struct. Does not actually allocate any memory.
	GFX_PixelBufferDestroy	Destroys a pixel buffer construct. If the buffer's pixels pointer is not zero this function will attempt to free it using the provided GFX_MemoryIntf memory interface.
	GFX_PixelBufferGet	Gets the value of the pixel that resides at the provided point in the given buffer.
	GFX_PixelBufferGet_Unsafe	Gets the value of the pixel that resides at the provided point in the given buffer. Like GFX_PixelBufferGet but performs no bounds checking.
	GFX_PixelBufferGetIndex	Interprets the pixel buffer as a table of indices and looks up a specific index at position 'idx'. Indices may be 1bpp, 4bpp, or 8bpp in size and are indicated by the color mode of the pixel buffer.
	GFX_PixelBufferOffsetGet	Gets the offset address of the pixel that resides at the provided point in the given buffer.
	GFX_PixelBufferOffsetGet_Unsafe	Gets the offset address of the pixel that resides at the provided point in the given buffer. Similar to GFX_PixelBufferOffsetGet but performs no bounds checking.
	GFX_PixelBufferSet	Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer.
	GFX_PixelBufferSet_Unsafe	Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer. Like GFX_PixelBufferSet but performs no bounds checking.
	GFX_RectClip	Clips a rectangle to the space of another rectangle. The result rectangle is a rectangle that will fit inside both of the given rectangles.
	GFX_RectContainsPoint	Determines if a point is inside a rectangle.
	GFX_RectContainsRect	Determines if a rectangle is completely inside another rectangle. Still returns true if the edges are touching.

	GFX_RectIntersects	Determines if two rectangles are intersecting
	GFX_ScaleInteger	Scales an integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.
	createDefaultMemIntf	internal use only
	GFX_ColorBilerp	Calculates bilinear interpolation between four colors
	GFX_ColorBlend_RGBA_8888	Blends two RGBA8888 colors together using their alpha channel values.
	GFX_ColorChannelBlue	Used for getting the blue color channel of a given color value.
	GFX_DivideRounding	This is function GFX_DivideRounding.
	GFX_DrawBlit	Blits a buffer of pixels into the frame buffer.
	GFX_DrawCircle	Draws a circle from using the specified dimensions and the current draw state.
	GFX_DrawLine	Draws a line from (x1,y1) to (x2,y2) using the current draw state.
	GFX_DrawPixel	Sets the pixel at X and Y using the current draw state.
	GFX_DrawRect	Draws a rectangle using the specified dimensions and the current draw state.
	GFX_DrawStretchBlit	Blits a buffer of pixels into the frame buffer.
	GFX_LayerFromOrientedSpace	Transforms a layer oriented space to screen space.
	GFX_LayerPointFromOrientedSpace	Transforms a point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerPointToOrientedSpace	Transforms a point from screen space to oriented space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerRectFromOrientedSpace	Transforms a layer point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerRectToOrientedSpace	Transforms a rectangle from screen space to oriented space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerToOrientedSpace	Transforms a layer from screen space to oriented space.
	GFX_PercentOfDec	This is function GFX_PercentOfDec.
	GFX_RectClipAdj	This is function GFX_RectClipAdj.
	GFX_RectFromPoints	This is function GFX_RectFromPoints.
	GFX_RectSplit	This is function GFX_RectSplit.
	GFX_RectToPoints	This is function GFX_RectToPoints.
	GFX_UtilMirrorPoint	Reorients a point to a given mirrored orientation.
	GFX_UtilOrientPoint	Reorients a point to a given orthogonal orientation.
	GFX_UtilPointFromOrientedSpace	Transforms a point from an oriented rectangle space to an outer space given a display orientation and a mirroring setting.
	GFX_UtilPointToOrientedSpace	Transforms a point to an oriented rectangle space to an outer space given a display orientation and a mirroring setting.
	GFX_UtilSizeFromOrientedSpace	Transforms a size tuple from oriented space to screen space
	GFX_UtilSizeToOrientedSpace	Transforms a size tuple from screen space to oriented space
	GFX_UtilSortPointsX	Sorts two points in the X axis
	GFX_UtilSortPointsY	Sorts two points in the Y axis
	GFX_RectCombine	Combines the area of two rectangles into a single rectangle.
	GFX_DrawDirectBlit	Blits a buffer of pixels into the frame buffer without performing per-pixel operations on the data.
	GFX_RectCompare	This is function GFX_RectCompare.
	GFX_RectsAreSimilar	This is function GFX_RectsAreSimilar.
	GFX_ScaleIntegerSigned	Scales a signed integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.

b) Data Types and Constants

	Name	Description
	GFX_AntialiasMode_t	Enables anti-aliased drawing hint
	GFX_BitsPerPixel_t	List of available bits-per-pixel sizes.
	GFX_BlendMode_t	Blend mode masks
	GFX_BufferSelection_t	Buffer selector used when querying layers for certain buffer states.
	GFX_BufferState_t	Frame buffer memory states
	GFX_ColorMask_t	Maskable list of color valies.
	GFX_ColorMode_t	List of available color modes.
	GFX_ColorModeInfo_t	Struct that provides information about a color mode.

	GFX_ColorName_t	Color name reference table
	GFX_Context_t	An instance of the hardware abstraction layer.
	GFX_DisplayInfo_t	Describes a graphical display device.
	GFX_DrawMode_t	Gradient draw modes.
	GFX_DrawState_t	A list of drawing hints for shape drawing algorithms
	GFX_DriverInfo_t	A driver description structure.
	GFX_Flag_t	Hardware abstraction state interface flags. See gfx.h for a comprehensive description.
	GFX_FrameBuffer_t	A frame buffer is a wrapper around a pixel buffer construct that is used by display drivers to manage frame buffers.
	GFX_HAL_t	Hardware Abstraction Function Table * • This is the core hardware abtraction table that makes everything work. Drivers are • expected to reroute the functionality of this table to hardware specific implementations • to provide accelerated performance and features.
	GFX_Layer_t	Layers describe basic display drawing areas and are meant to map directly to graphics controller hardware layers.
	GFX_MemoryIntf_t	Custom memory manager interface.
	GFX_Orientation_t	Orthogonal orientation settings.
	GFX_PixelBuffer_t	A pixel buffer is a wrapper around a basic data pointer. A pixel buffer has a color mode, a pixel count, a rectangular dimension, a pixel count, and a lenght in bytes.
	GFX_Point_t	A two dimensional Cartesian point.
	GFX_Rect_t	A rectangle definition.
	GFX_Size_t	A two dimensional indication of size. Values are signed but should never be negative.
	begin_FnPtr	This is type begin_FnPtr.
	boolGet_FnPtr	This is type boolGet_FnPtr.
	boolSet_FnPtr	This is type boolSet_FnPtr.
	brightnessGet_FnPtr	This is type brightnessGet_FnPtr.
	brightnessRangeGet_FnPtr	This is type brightnessRangeGet_FnPtr.
	brightnessSet_FnPtr	This is type brightnessSet_FnPtr.
	colorModeGet_FnPtr	This is type colorModeGet_FnPtr.
	colorModeSet_FnPtr	This is type colorModeSet_FnPtr.
	destroy_FnPtr	This is type destroy_FnPtr.
	drawAlphaValueGet_FnPtr	This is type drawAlphaValueGet_FnPtr.
	drawAlphaValueSet_FnPtr	This is type drawAlphaValueSet_FnPtr.
	drawBlendModeGet_FnPtr	This is type drawBlendModeGet_FnPtr.
	drawBlendModeSet_FnPtr	This is type drawBlendModeSet_FnPtr.
	drawBlit_FnPtr	This is type drawBlit_FnPtr.
	drawCircle_FnPtr	This is type drawCircle_FnPtr.
	drawClipRectGet_FnPtr	This is type drawClipRectGet_FnPtr.
	drawClipRectSet_FnPtr	This is type drawClipRectSet_FnPtr.
	drawColorGet_FnPtr	This is type drawColorGet_FnPtr.
	drawColorSet_FnPtr	This is type drawColorSet_FnPtr.
	drawGradientColorGet_FnPtr	This is type drawGradientColorGet_FnPtr.
	drawGradientColorSet_FnPtr	This is type drawGradientColorSet_FnPtr.
	drawLine_FnPtr	This is type drawLine_FnPtr.
	drawLock_FnPtr	This is type drawLock_FnPtr.
	drawMaskValueGet_FnPtr	This is type drawMaskValueGet_FnPtr.
	drawMaskValueSet_FnPtr	This is type drawMaskValueSet_FnPtr.
	drawModeGet_FnPtr	This is type drawModeGet_FnPtr.
	drawModeSet_FnPtr	This is type drawModeSet_FnPtr.
	drawPaletteGet_FnPtr	This is type drawPaletteGet_FnPtr.
	drawPaletteSet_FnPtr	This is type drawPaletteSet_FnPtr.
	drawPixel_FnPtr	This is type drawPixel_FnPtr.
	drawRect_FnPtr	This is type drawRect_FnPtr.
	drawThicknessGet_FnPtr	This is type drawThicknessGet_FnPtr.
	drawThicknessSet_FnPtr	This is type drawThicknessSet_FnPtr.
	drawUnlock_FnPtr	This is type drawUnlock_FnPtr.

	GFX_AntialiasMode	Enables anti-aliased drawing hint
	GFX_BitsPerPixel	List of available bits-per-pixel sizes.
	GFX_BlendMode	Blend mode masks
	GFX_Bool	This is type GFX_Bool.
	GFX_Buffer	This is type GFX_Buffer.
	GFX_BufferSelection	Buffer selector used when querying layers for certain buffer states.
	GFX_BufferState	Frame buffer memory states
	GFX_Calloc_FnPtr	Simple wrapper around the standard calloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Color	This is type GFX_Color.
	GFX_ColorMask	Maskable list of color valies.
	GFX_ColorMode	List of available color modes.
	GFX_ColorModeInfo	Struct that provides information about a color mode.
	GFX_ColorName	Color name reference table
	GFX_Context	An instance of the hardware abstraction layer.
	GFX_Display	This is type GFX_Display.
	GFX_DisplayInfo	Describes a graphical display device.
	GFX_DrawMode	Gradient draw modes.
	GFX_DrawState	A list of drawing hints for shape drawing algorithms
	GFX_Driver	This is type GFX_Driver.
	GFX_DriverInfo	A driver description structure.
	GFX_Flag	Hardware abstraction state interface flags. See gfx.h for a comprehensive description.
	GFX_FrameBuffer	A frame buffer is a wrapper around a pixel buffer construct that is used by display drivers to manage frame buffers.
	GFX_Free_FnPtr	Simple wrapper around the standard free function pointer. Used for redirecting memory free to other memory management systems.
	GFX_HAL	Hardware Abstraction Function Table * • This is the core hardware abtraction table that makes everything work. Drivers are • expected to reroute the functionality of this table to hardware specific implementations • to provide accelerated performance and features.
	GFX_Handle	This is type GFX_Handle.
	GFX_Layer	Layers describe basic display drawing areas and are meant to map directly to graphics controller hardware layers.
	GFX_Malloc_FnPtr	Simple wrapper around the standard malloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Memcpy_FnPtr	Simple wrapper around the standard memcpy function pointer. Used for redirecting memcpy to other memory management systems.
	GFX_MemoryIntf	Custom memory manager interface.
	GFX_Memset_FnPtr	Simple wrapper around the standard memset function pointer. Used for redirecting memset to other memory management systems.
	GFX_Orientation	Orthogonal orientation settings.
	GFX_PixelBuffer	A pixel buffer is a wrapper around a basic data pointer. A pixel buffer has a color mode, a pixel count, a rectangular dimension, a pixel count, and a lenght in bytes.
	end_FnPtr	This is type end_FnPtr.
	GFX_Processor	This is type GFX_Processor.
	GFX_Realloc_FnPtr	Simple wrapper around the standard realloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Result	This is type GFX_Result.
	GFX_Size	A two dimensional indication of size. Values are signed but should never be negative.
	GFX_SyncCallback_FnPtr	This is type GFX_SyncCallback_FnPtr.
	layerActiveGet_FnPtr	This is type layerActiveGet_FnPtr.
	layerActiveSet_FnPtr	This is type layerActiveSet_FnPtr.
	layerAlphaAmountGet_FnPtr	This is type layerAlphaAmountGet_FnPtr.
	layerAlphaAmountSet_FnPtr	This is type layerAlphaAmountSet_FnPtr.
	layerBufferAddressGet_FnPtr	This is type layerBufferAddressGet_FnPtr.
	layerBufferAddressSet_FnPtr	This is type layerBufferAddressSet_FnPtr.
	layerBufferAllocate_FnPtr	This is type layerBufferAllocate_FnPtr.

	layerBufferCoherentGet_FnPtr	This is type layerBufferCoherentGet_FnPtr.
	layerBufferCoherentSet_FnPtr	This is type layerBufferCoherentSet_FnPtr.
	layerBufferCountGet_FnPtr	This is type layerBufferCountGet_FnPtr.
	layerBufferCountSet_FnPtr	This is type layerBufferCountSet_FnPtr.
	layerBufferFree_FnPtr	This is type layerBufferFree_FnPtr.
	layerBufferIsAllocated_FnPtr	This is type layerBufferIsAllocated_FnPtr.
	layerMaskColorGet_FnPtr	This is type layerMaskColorGet_FnPtr.
	layerMaskColorSet_FnPtr	This is type layerMaskColorSet_FnPtr.
	layerPositionGet_FnPtr	This is type layerPositionGet_FnPtr.
	layerPositionSet_FnPtr	This is type layerPositionSet_FnPtr.
	layerSizeGet_FnPtr	This is type layerSizeGet_FnPtr.
	layerSizeSet_FnPtr	This is type layerSizeSet_FnPtr.
	layerSwapped_FnPtr	This is type layerSwapped_FnPtr.
	orientationGet_FnPtr	This is type orientationGet_FnPtr.
	orientationSet_FnPtr	This is type orientationSet_FnPtr.
	pixelGet_FnPtr	This is type pixelGet_FnPtr.
	pixelSet_FnPtr	This is type pixelSet_FnPtr.
	syncCallbackGet_FnPtr	This is type syncCallbackGet_FnPtr.
	syncCallbackSet_FnPtr	This is type syncCallbackSet_FnPtr.
	syncCallbackSt_FnPtr	This is type syncCallbackSt_FnPtr.
	update_FnPtr	This is type update_FnPtr.
	GFX_ColorInfo	This is variable GFX_ColorInfo.
	GFX_Rect_Zero	This is variable GFX_Rect_Zero.
	AGBA_8888_ALPHA_MASK	This is macro AGBA_8888_ALPHA_MASK.
	AGBA_8888_BLUE_MASK	This is macro AGBA_8888_BLUE_MASK.
	AGBA_8888_GREEN_MASK	This is macro AGBA_8888_GREEN_MASK.
	AGBA_8888_RED_MASK	This is macro AGBA_8888_RED_MASK.
	GFX_ANTIALIAS_MODE_COUNT	This is macro GFX_ANTIALIAS_MODE_COUNT.
	GFX_COLOR_MAX_SIZE	This is macro GFX_COLOR_MAX_SIZE.
	GFX_COLOR_MODE_COUNT	This is macro GFX_COLOR_MODE_COUNT.
	GFX_COLOR_MODE_IS_ALPHA	This is macro GFX_COLOR_MODE_IS_ALPHA.
	GFX_COLOR_MODE_IS_INDEX	This is macro GFX_COLOR_MODE_IS_INDEX.
	GFX_COLOR_MODE_IS_PIXEL	This is macro GFX_COLOR_MODE_IS_PIXEL.
	GFX_COLOR_MODE_LAST_COLOR	This is macro GFX_COLOR_MODE_LAST_COLOR.
	GFX_DRAW_MODE_COUNT	This is macro GFX_DRAW_MODE_COUNT.
	GFX_FAILURE	This is macro GFX_FAILURE.
	GFX_FALSE	This is macro GFX_FALSE.
	GFX_MAX_BUFFER_COUNT	This is macro GFX_MAX_BUFFER_COUNT.
	GFX_NULL	This is macro GFX_NULL.
	GFX_NUM_FLAGS	This is macro GFX_NUM_FLAGS.
	GFX_SUCCESS	This is macro GFX_SUCCESS.
	GFX_TRUE	This is macro GFX_TRUE.
	GFX_UNSUPPORTED	This is macro GFX_UNSUPPORTED.
	RGB_2_BITS	This is macro RGB_2_BITS.
	RGB_3_BITS	This is macro RGB_3_BITS.
	RGB_332_BLUE_MASK	This is macro RGB_332_BLUE_MASK.
	RGB_332_GREEN_MASK	This is macro RGB_332_GREEN_MASK.
	RGB_332_RED_MASK	This is macro RGB_332_RED_MASK.
	RGB_5_BITS	This is macro RGB_5_BITS.
	RGB_565_BLUE_MASK	This is macro RGB_565_BLUE_MASK.
	RGB_565_GREEN_MASK	This is macro RGB_565_GREEN_MASK.
	RGB_565_RED_MASK	This is macro RGB_565_RED_MASK.
	RGB_6_BITS	This is macro RGB_6_BITS.
	RGB_8_BITS	This is macro RGB_8_BITS.
	RGB_888_BLUE_MASK	This is macro RGB_888_BLUE_MASK.

	RGB_888_GREEN_MASK	This is macro RGB_888_GREEN_MASK.
	RGB_888_RED_MASK	This is macro RGB_888_RED_MASK.
	RGBA_5551_ALPHA_MASK	This is macro RGBA_5551_ALPHA_MASK.
	RGBA_5551_BLUE_MASK	This is macro RGBA_5551_BLUE_MASK.
	RGBA_5551_GREEN_MASK	This is macro RGBA_5551_GREEN_MASK.
	RGBA_5551_RED_MASK	This is macro RGBA_5551_RED_MASK.
	RGBA_8888_ALPHA_MASK	This is macro RGBA_8888_ALPHA_MASK.
	RGBA_8888_BLUE_MASK	This is macro RGBA_8888_BLUE_MASK.
	RGBA_8888_GREEN_MASK	This is macro RGBA_8888_GREEN_MASK.
	RGBA_8888_RED_MASK	This is macro RGBA_8888_RED_MASK.
	initialize_FnPtr	This is type initialize_FnPtr.
	interrupt_FnPtr	GFX_DRAW_PIPELINE_ENABLED
	GFX_DrawPipeline_t	
	GFX_PipelineMode_t	Hardware draw path settings.
	GFX_ResizeMode_t	This is type GFX_ResizeMode.
	GFXU_ImageFlags_t	A list of flags describing an image asset
	blendColor_FnPtr	This is type blendColor_FnPtr.
	blendGetPoint_FnPtr	This is type blendGetPoint_FnPtr.
	drawPipelineModeGet_FnPtr	This is type drawPipelineModeGet_FnPtr.
	drawPipelineModeSet_FnPtr	This is type drawPipelineModeSet_FnPtr.
	drawResizeModeGet_FnPtr	This is type drawResizeModeGet_FnPtr.
	drawResizeModeSet_FnPtr	This is type drawResizeModeSet_FnPtr.
	drawStretchBlit_FnPtr	This is type drawStretchBlit_FnPtr.
	drawTargetGet_FnPtr	This is type drawTargetGet_FnPtr.
	drawTargetSet_FnPtr	This is type drawTargetSet_FnPtr.
	GFX_DrawPipeline	
	GFX_PipelineMode	Hardware draw path settings.
	GFX_ResizeMode	This is type GFX_ResizeMode.
	GFXU_ImageFlags	A list of flags describing an image asset
	layerSwapPending_FnPtr	This is type layerSwapPending_FnPtr.
	maskColor_FnPtr	This is type maskColor_FnPtr.
	mirrorPoint_FnPtr	This is type mirrorPoint_FnPtr.
	orientPoint_FnPtr	This is type orientPoint_FnPtr.
	pixelGetArray_FnPtr	This is type pixelGetArray_FnPtr.
	GFX_PIPELINE_MODE_COUNT	This is macro GFX_PIPELINE_MODE_COUNT.
	GFX_RESIZE_MODE_COUNT	This is macro GFX_RESIZE_MODE_COUNT.
	GFX_ASSERT	This is macro GFX_ASSERT.
	GFX_GLOBAL_PALETTE_SIZE	This is macro GFX_GLOBAL_PALETTE_SIZE.
	GFX_GlobalPalette	This is type GFX_GlobalPalette.
	globalPaletteGet_FnPtr	This is type globalPaletteGet_FnPtr.
	globalPaletteSet_FnPtr	This is type globalPaletteSet_FnPtr.
	layerEffectSet_FnPtr	This is type layerEffectSet_FnPtr.
	GFX_DEPRECATED	This is macro GFX_DEPRECATED.

Description

This section describes the Aria User Interface Library Hardware Abstraction Layer interface.

a) Functions

GFX_AbsoluteValue Function

Calculates the absolute value of a signed integer.

File

gfx_math.h

C

```
LIB_EXPORT uint32_t GFX_AbsoluteValue(int32_t val);
```

Returns

uint32_t - the absolute value

Parameters

Parameters	Description
val	the number to consider

Function

uint32_t GFX_AbsoluteValue(int32_t val);

GFX_ActiveContext Function

Gets the current set active HAL context.

File

gfx_context.h

C

```
LIB_EXPORT GFX_Context* GFX_ActiveContext();
```

Returns

GFX_Context* - the active context or NULL

Function

GFX_Context* GFX_ActiveContext(void)

GFX_Clampf Function

Clamps a float between a min and max

File

gfx_math.h

C

```
LIB_EXPORT float GFX_Clampf(float min, float max, float f);
```

Returns

float - the clamped value

Parameters

Parameters	Description
min	the minimum value
max	the maximum value
i	the float to clamp

Function

float GFX_Clampi(float min, float max, float i);

GFX_Clampi Function

Clamps an integer between a min and max

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_Clampi(int32_t min, int32_t max, int32_t i);
```

Returns

int32_t - the clamped value

Parameters

Parameters	Description
min	the minimum value
max	the maximum value
i	the number to clamp

Function

int32_t GFX_Clampi(int32_t min, int32_t max, int32_t i);

GFX_ColorChannelAlpha Function

Used for getting the alpha color channel of a given color value.

File

gfx_color.h

C

```
LIB_EXPORT uint32_t GFX_ColorChannelAlpha(GFX_Color clr, GFX_ColorMode mode);
```

Returns

uint32_t - the alpha color channel

Parameters

Parameters	Description
GFX_Color	the source color value
GFX_ColorMode	the source color mode

Function

uint32_t GFX_ColorChannelAlpha(GFX_Color clr, GFX_ColorMode mode)

GFX_ColorChannelGreen Function

Used for getting the green color channel of a given color value.

File

gfx_color.h

C

```
LIB_EXPORT uint32_t GFX_ColorChannelGreen(GFX_Color clr, GFX_ColorMode mode);
```

Returns

uint32_t - the green color channel

Parameters

Parameters	Description
GFX_Color	the source color value
GFX_ColorMode	the source color mode

Function

uint32_t GFX_ColorChannelGreen(GFX_Color clr, GFX_ColorMode mode)

GFX_ColorChannelRed Function

Used for getting the red color channel of a given color value.

File

gfx_color.h

C

```
LIB_EXPORT uint32_t GFX_ColorChannelRed(GFX_Color clr, GFX_ColorMode mode);
```

Returns

uint32_t - the red color channel

Parameters

Parameters	Description
GFX_Color	the source color value
GFX_ColorMode	the source color mode

Function

uint32_t GFX_ColorChannelRed(GFX_Color clr, GFX_ColorMode mode)

GFX_ColorConvert Function

Converts a color value from one mode to another

File

gfx_color.h

C

```
LIB_EXPORT GFX_Color GFX_ColorConvert(GFX_ColorMode mode_in, GFX_ColorMode mode_out, GFX_Color color);
```

Returns

GFX_Color - the result color

Parameters

Parameters	Description
GFX_ColorMode	the input color mode the output color mode
GFX_Color	the source color

Function

GFX_Color GFX_ColorConvert(GFX_ColorMode mode_in,
GFX_ColorMode mode_out,
GFX_Color color)

GFX_ColorLerp Function

Linear interpolation between two colors

File

gfx_color.h

C

```
LIB_EXPORT GFX_Color GFX_ColorLerp(GFX_Color l, GFX_Color r, uint32_t percent, GFX_ColorMode mode);
```

Returns

GFX_Color - the result color

Parameters

Parameters	Description
GFX_Color	first color input second color input
uint32_t	percentage of interpolation [0-100]
GFX_ColorMode	input color mode

Function

GFX_Color GFX_ColorLerp(GFX_Color l,
GFX_Color r,

uint32_t percent,
GFX_ColorMode mode)

GFX_ColorModeInfoGet Function

File

gfx_color.h

C

```
LIB_EXPORT GFX_ColorModeInfo GFX_ColorModeInfoGet(GFX_ColorMode mode);
```

Section

Routines

GFX_ColorValue Function

Used for getting a color value by name.

File

gfx_color.h

C

```
LIB_EXPORT GFX_Color GFX_ColorValue(GFX_ColorMode mode, GFX_ColorName name);
```

Returns

GFX_Color - the color value of the given name in the specified format

Parameters

Parameters	Description
GFX_ColorMode	the color mode for the return type
GFX_ColorName	the name of the color to retrieve

Function

GFX_Color GFX_ColorValue(GFX_ColorMode mode, GFX_ColorName name)

GFX_ContextActiveSet Function

Sets the active context

File

gfx_context.h

C

```
void GFX_ContextActiveSet(GFX_Context* const context);
```

Parameters

Parameters	Description
GFX_Context*	the new active context or NULL

Function

void GFX_ContextActiveSet(GFX_Context* const context)

GFX_LayerReadBuffer Function

Gets the pointer to the layer's current read pixel buffer.

File

gfx_layer.h

C

```
GFX_PixelBuffer* GFX_LayerReadBuffer(GFX_Layer* layer);
```

Returns

GFX_PixelBuffer* - the pointer to the read pixel buffer

Parameters

Parameters	Description
GFX_Layer*	the pointer to the layer

Function

GFX_PixelBuffer* GFX_LayerReadBuffer(GFX_Layer* layer)

GFX_LayerRotate Function

Swaps the width and height dimensions of a layer. Can be used for run-time display orientation

File

gfx_layer.h

C

```
void GFX_LayerRotate(GFX_Layer* layer);
```

Parameters

Parameters	Description
GFX_Layer*	the layer to operate on

Function

void GFX_LayerRotate(GFX_Layer* layer)

GFX_LayerSwap Function

Performs a swap operation on the given layer. This advances the pointers of layer's buffer chain. The current write buffer becomes the new read buffer and a new buffer is chosen as the new write buffer. Has no effect in single buffer environments.

File

gfx_layer.h

C

```
void GFX_LayerSwap(GFX_Layer* layer);
```

Parameters

Parameters	Description
GFX_Layer*	the layer to operate on

Function

void GFX_LayerSwap(GFX_Layer* layer)

GFX_LayerWriteBuffer Function

Gets the pointer to the layer's current write pixel buffer.

File

gfx_layer.h

C

```
GFX_PixelBuffer* GFX_LayerWriteBuffer(GFX_Layer* layer);
```

Returns

GFX_PixelBuffer* - the pointer to the write pixel buffer

Parameters

Parameters	Description
GFX_Layer*	the pointer to the layer

Function

GFX_PixelBuffer* GFX_LayerWriteBuffer(GFX_Layer* layer)

GFX_Lerp Function

Performs a linear interpolation of an integer based on a percentage between two signed points.

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_Lerp(int32_t x, int32_t y, uint32_t per);
```

Returns

int32_t - the interpolated value

Parameters

Parameters	Description
x	the first point to consider
y	the second point to consider
per	the percentage of interpolation

Function

int32_t GFX_Lerp(int32_t x, int32_t y, uint32_t per);

GFX_Maxf Function

Returns the larger of two floats.

File

gfx_math.h

C

```
LIB_EXPORT float GFX_Maxf(float l, float r);
```

Returns

float - the larger of the two floats

Parameters

Parameters	Description
l	the first float to test
r	the second float to test

Function

float GFX_Maxf(float l, float r);

GFX_Maxi Function

Returns the larger of two integers.

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_Maxi(int32_t l, int32_t r);
```

Returns

int32_t - the larger of the two numbers

Parameters

Parameters	Description
l	the first number to test
r	the second number to test

Function

int32_t GFX_Maxi(int32_t l, int32_t r);

GFX_Minf Function

Returns the smaller of two floats.

File

gfx_math.h

C

```
LIB_EXPORT float GFX_Minf(float l, float r);
```

Returns

float - the smaller of the two floats

Parameters

Parameters	Description
l	the first float to test
r	the second float to test

Function

float GFX_Minf(float l, float r);

GFX_Mini Function

Returns the smaller of two integers.

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_Mini(int32_t l, int32_t r);
```

Returns

int32_t - the smaller of the two numbers

Parameters

Parameters	Description
l	the first number to test
r	the second number to test

Function

int32_t GFX_Mini(int32_t l, int32_t r);

GFX_Percent Function

Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed.

The result is the decimal percentage multiplied by 100.

File

gfx_math.h

C

```
LIB_EXPORT uint32_t GFX_Percent(uint32_t l, uint32_t r);
```

Returns

uint32_t - the percentage represented as a whole number

Parameters

Parameters	Description
l	the first number of the equation
r	the second number of the equation

Function

uint32_t GFX_Percent(uint32_t l, uint32_t r);

GFX_PercentOf Function

Calculates the percentage of a number. Returns a whole number with no decimal component.

File

gfx_math.h

C

```
LIB_EXPORT uint32_t GFX_PercentOf(uint32_t num, uint32_t percent);
```

Returns

uint32_t - the resultant percentage of the number

Parameters

Parameters	Description
num	the number to consider
percent	the percentage to apply

Function

uint32_t GFX_PercentOf(uint32_t l, uint32_t percent);

GFX_PercentWholeRounded Function

Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed.

The difference between this and GFX_Percent is that the decimal portion of the whole number is rounded off.

File

gfx_math.h

C

```
LIB_EXPORT uint32_t GFX_PercentWholeRounded(uint32_t l, uint32_t r);
```

Returns

uint32_t - the percentage represented as a whole number

Parameters

Parameters	Description
l	the first number of the equation
r	the second number of the equation

Function

uint32_t GFX_PercentWholeRounded(uint32_t l, uint32_t r);

GFX_PixelBufferAreaFill Function

Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaFill(const GFX_PixelBuffer* const buffer, const GFX_Rect* const
rect, const GFX_Color color);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the buffer to manipulate
const GFX_Rect* const rect	the rectangle of the buffer to fill
const GFX_Color color	the color to use for the fill operation

Function

GFX_Result GFX_PixelBufferAreaFill(const GFX_PixelBuffer* const buffer,
const GFX_Rect* const rect,
const GFX_Color color)

GFX_PixelBufferAreaFill_Unsafe Function

Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer. Like GFX_PixelBufferAreaFill but performs no bounds checking.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaFill_Unsafe(const GFX_PixelBuffer* const buffer, const GFX_Rect*
const rect, const GFX_Color color);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the buffer to manipulate
const GFX_Rect* const rect	the rectangle of the buffer to fill
const GFX_Color color	the color to use for the fill operation

Function

GFX_Result GFX_PixelBufferAreaFill_Unsafe(const GFX_PixelBuffer* const buffer,
const GFX_Rect* const rect,
const GFX_Color color)

GFX_PixelBufferAreaGet Function

Extracts a rectanglar section of pixels from a pixel buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaGet(const GFX_PixelBuffer* const buffer, const GFX_Rect* const
rect, GFX_MemoryIntf* mem_intf, GFX_PixelBuffer* out);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the source buffer
const GFX_Rect* rect	the area to extract
GFX_MemoryIntf*	the memory interface to use for memory operations
GFX_PixelBuffer*	the resultant pixel buffer

Function

GFX_Result GFX_PixelBufferAreaGet(const GFX_PixelBuffer* const buffer,
const GFX_Rect* const rect,
GFX_MemoryIntf* mem_intf,
GFX_PixelBuffer* out)

GFX_PixelBufferAreaGet_Unsafe Function

Extracts a rectanglar section of pixels from a pixel buffer. Like GFX_PixelBufferAreaGet but performs no clipping between the rectangles of the extract area and the source buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaGet_Unsafe(const GFX_PixelBuffer* const buffer, const GFX_Rect*
const rect, GFX_MemoryIntf* mem_intf, GFX_PixelBuffer* out);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the source buffer
const GFX_Rect* rect	the area to extract
GFX_MemoryIntf*	the memory interface to use for memory operations
GFX_PixelBuffer*	the resultant pixel buffer

Function

GFX_Result GFX_PixelBufferAreaGet_Unsafe(const GFX_PixelBuffer* const buffer,
const GFX_Rect* const rect,
GFX_MemoryIntf* mem_intf,
GFX_PixelBuffer* out)

GFX_PixelBufferAreaSet Function

Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaSet(const GFX_PixelBuffer* const source, const GFX_Rect* const
source_rect, const GFX_PixelBuffer* const dest, const GFX_Point* const pnt, GFX_MemoryIntf* mem_intf);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const source	the source buffer

const GFX_Rect* const source_rect	the rectangle of the source buffer to use
const GFX_PixelBuffer* const dest	the destination buffer to copy to
const GFX_Point* const pnt	the location of the destination to copy to
GFX_MemoryIntf*	the memory interface to use for memory operations

Function

GFX_Result GFX_PixelBufferAreaSet(const GFX_PixelBuffer* const source,
const GFX_Rect* const source_rect,
const GFX_PixelBuffer* const dest,
const GFX_Point* const pnt,
GFX_MemoryIntf* mem_intf)

GFX_PixelBufferAreaSet_Unsafe Function

Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation. Like GFX_PixelBufferAreaSet but performs no bounds checking.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferAreaSet_Unsafe(const GFX_PixelBuffer* const source, const GFX_Rect*
const source_rect, const GFX_PixelBuffer* const dest, const GFX_Point* const pnt, GFX_MemoryIntf* mem_intf);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const source	the source buffer
const GFX_Rect* const source_rect	the rectangle of the source buffer to use
const GFX_PixelBuffer* const dest	the destination buffer to copy to
const GFX_Point* const pnt	the location of the destination to copy to
GFX_MemoryIntf*	the memory interface to use for memory operations

Function

GFX_Result GFX_PixelBufferAreaSet_Unsafe(const GFX_PixelBuffer* const source,
const GFX_Rect* const source_rect,
const GFX_PixelBuffer* const dest,
const GFX_Point* const pnt,
GFX_MemoryIntf* mem_intf)

GFX_PixelBufferClipRect Function

Clips a rectangle against a pixel buffer. The result is guaranteed to fit inside the buffer's area.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferClipRect(const GFX_PixelBuffer* const buffer, const GFX_Rect* const
rect, GFX_Rect* result);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the source buffer

const GFX_Rect* const	the rectangle to analyze
GFX_Rect* result	the clipped rectangle

Function

GFX_Result GFX_PixelBufferClipRect(const GFX_PixelBuffer* const buffer,
const GFX_Rect* const rect,
GFX_Rect* result)

GFX_PixelBufferConvert Function

Duplicates a pixel buffer and converts the copy to another color mode.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferConvert(const GFX_PixelBuffer* const source, const GFX_ColorMode
result_mode, GFX_MemoryIntf* mem_intf, GFX_PixelBuffer* result);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const source	the source buffer
const GFX_ColorMode result_mode	the desired color mode
GFX_MemoryIntf*	the memory interface to use for memory operations
GFX_PixelBuffer*	the resultant pixel buffer

Function

GFX_Result GFX_PixelBufferConvert(const GFX_PixelBuffer* const source,
const GFX_ColorMode result_mode,
GFX_MemoryIntf* mem_intf,
GFX_PixelBuffer* result)

GFX_PixelBufferCopy Function

Creates a copy of the input buffer. If the input buffer's pixel data is not null then the data will be duplicated for the result buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferCopy(const GFX_PixelBuffer* const buffer, GFX_MemoryIntf* mem_intf,
GFX_PixelBuffer* result);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the source buffer
GFX_MemoryIntf*	the memory interface to use for memory operations
GFX_PixelBuffer*	the result buffer

Function

GFX_Result GFX_PixelBufferCopy(const GFX_PixelBuffer* const buffer,
GFX_MemoryIntf* mem_intf,
GFX_PixelBuffer* result)

GFX_PixelBufferCreate Function

Initializes a pixel buffer struct. Does not actually allocate any memory.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferCreate(const int32_t width, const int32_t height, const GFX_ColorMode
mode, const void* const address, GFX_PixelBuffer* buffer);
```

Returns

GFX_Result

Parameters

Parameters	Description
const int32_t	the width of the buffer the height of the buffer
const GFX_ColorMode	the color mode of the buffer
const void*	the data addres of the buffer (may be NULL)
GFX_PixelBuffer*	pointer of the pixel buffer buffer to initialize

Function

GFX_Result GFX_PixelBufferCreate(const int32_t width,
const int32_t height,
const GFX_ColorMode mode,
const void* const address,
GFX_PixelBuffer* buffer)

GFX_PixelBufferDestroy Function

Destroys a pixel buffer construct. If the buffer's pixels pointer is not zero this function will attempt to free it using the provided GFX_MemoryIntf memory interface.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferDestroy(GFX_PixelBuffer* const buffer, GFX_MemoryIntf* mem_intf);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFX_PixelBuffer*	the buffer to destroy
GFX_MemoryIntf*	the memory interface to reference for free()

Function

GFX_Result GFX_PixelBufferDestroy(GFX_PixelBuffer* const buffer,
GFX_MemoryIntf* mem_intf)

GFX_PixelBufferGet Function

Gets the value of the pixel that resides at the provided point in the given buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Color GFX_PixelBufferGet(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt);
```

Returns

GFX_Color - the value of the pixel at the point in the source buffer

Parameters

Parameters	Description
const GFX_PixelBuffer*	the source buffer
const GFX_Point*	the point for which the offset should be calculated

Function

GFX_Color GFX_PixelBufferGet(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt)

GFX_PixelBufferGet_Unsafe Function

Gets the value of the pixel that resides at the provided point in the given buffer. Like GFX_PixelBufferGet but performs no bounds checking.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Color GFX_PixelBufferGet_Unsafe(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt);
```

Returns

GFX_Color - the value of the pixel at the point in the source buffer

Parameters

Parameters	Description
const GFX_PixelBuffer*	the source buffer
const GFX_Point*	the point for which the offset should be calculated

Function

GFX_Color GFX_PixelBufferGet_Unsafe(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt)

GFX_PixelBufferGetIndex Function

Interprets the pixel buffer as a table of indices and looks up a specific index at position 'idx'. Indices may be 1bpp, 4bpp, or 8bpp in size and are indicated by the color mode of the pixel buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Color GFX_PixelBufferGetIndex(const GFX_PixelBuffer* const buffer, const int32_t idx);
```

Returns

GFX_Color - the resultant value that was retrieved

Parameters

Parameters	Description
const GFX_PixelBuffer* const	the input buffer
const int32_t	the index to retrieve

Function

GFX_Color GFX_PixelBufferGetIndex(const GFX_PixelBuffer* const buffer,
const int32_t idx)

GFX_PixelBufferOffsetGet Function

Gets the offset address of the pixel that resides at the provided point in the given buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Buffer GFX_PixelBufferOffsetGet(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt);
```

Returns

GFX_Buffer - the pointer to the offset point in the source buffer

Parameters

Parameters	Description
const GFX_PixelBuffer*	the source buffer
const GFX_Point*	the point for which the offset should be calculated

Function

GFX_Buffer GFX_PixelBufferOffsetGet(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt)

GFX_PixelBufferOffsetGet_Unsafe Function

Gets the offset address of the pixel that resides at the provided point in the given buffer. Similar to GFX_PixelBufferOffsetGet but performs no bounds checking.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Buffer GFX_PixelBufferOffsetGet_Unsafe(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt);
```

Returns

GFX_Buffer - the pointer to the offset point in the source buffer

Parameters

Parameters	Description
const GFX_PixelBuffer*	the source buffer
const GFX_Point*	the point for which the offset should be calculated

Function

GFX_Buffer GFX_PixelBufferOffsetGet_Unsafe(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt)

GFX_PixelBufferSet Function

Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferSet(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt, GFX_Color color);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the buffer to operate on
const GFX_Point* const	the location of the pixel to set
GFX_Color	the color to set the pixel to. must be the same format as the buffer

Function

GFX_Result GFX_PixelBufferSet(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt,
GFX_Color color)

GFX_PixelBufferSet_Unsafe Function

Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer. Like GFX_PixelBufferSet but performs no bounds checking.

File

gfx_pixel_buffer.h

C

```
LIB_EXPORT GFX_Result GFX_PixelBufferSet_Unsafe(const GFX_PixelBuffer* const buffer, const GFX_Point* const pnt, GFX_Color color);
```

Returns

GFX_Result

Parameters

Parameters	Description
const GFX_PixelBuffer* const buffer	the buffer to operate on
const GFX_Point* const	the location of the pixel to set
GFX_Color	the color to set the pixel to. must be the same format as the buffer

Function

GFX_Result GFX_PixelBufferSet_Unsafe(const GFX_PixelBuffer* const buffer,
const GFX_Point* const pnt,
GFX_Color color)

GFX_RectClip Function

Clips a rectangle to the space of another rectangle. The result rectangle is a rectangle that will fit inside both of the given rectangles.

File

gfx_rect.h

C

```
LIB_EXPORT void GFX_RectClip(const GFX_Rect* l_rect, const GFX_Rect* r_rect, GFX_Rect* result);
```

Returns

void

Remarks

result will equals l_rect if the rectangles aren't intersecting

Parameters

Parameters	Description
const GFX_Rect* l_rect	the subject rectangle
const GFX_Rect* r_rect	the object rectangle
GFX_Rect* result	the result rectangle

Function

void GFX_RectClip(const GFX_Rect* l_rect,
const GFX_Rect* r_rect,
GFX_Rect* result)

GFX_RectContainsPoint Function

Determines if a point is inside a rectangle.

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Bool GFX_RectContainsPoint(const GFX_Rect* rect, const GFX_Point* point);
```

Returns

GFX_Bool - GFX_TRUE if the point is inside the rectangle

Parameters

Parameters	Description
const GFX_Rect* rect	the rectangle to test
const GFX_Point* point	the point to use for the test

Function

GFX_Bool GFX_RectContainsPoint(const GFX_Rect* rect, const GFX_Point* point)

GFX_RectContainsRect Function

Determines if a rectangle is completely inside another rectangle. Still returns true if the edges are touching.

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Bool GFX_RectContainsRect(const GFX_Rect* l_rect, const GFX_Rect* r_rect);
```

Returns

GFX_Bool - returns GFX_TRUE if r_rect is completly inside l_rect

Parameters

Parameters	Description
const GFX_Rect* l_rect	the subject rectangle
const GFX_Rect* r_rect	the object rectangle

Function

GFX_Bool GFX_RectContainsRect(const GFX_Rect* l_rect, const GFX_Rect* r_rect)

GFX_RectIntersects Function

Determines if two rectangles are intersecting

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Bool GFX_RectIntersects(const GFX_Rect* l_rect, const GFX_Rect* r_rect);
```

Returns

GFX_Bool - returns GFX_TRUE if l_rect and r_rect are intersecting

Parameters

Parameters	Description
const GFX_Rect* l_rect	rectangle argument
const GFX_Rect* r_rect	rectangle argument

Function

GFX_Bool GFX_RectIntersects(const GFX_Rect* l_rect, const GFX_Rect* r_rect)

GFX_ScaleInteger Function

Scales an integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.

File

gfx_math.h

C

```
LIB_EXPORT uint32_t GFX_ScaleInteger(uint32_t num, uint32_t oldMax, uint32_t newMax);
```

Returns

uint32_t - the number as defined in the new number range

Parameters

Parameters	Description
num	the number to consider
oldMax	the old range maximum
newMax	the new range maximum

Function

uint32_t GFX_ScaleInteger(uint32_t num, uint32_t oldMax, uint32_t newMax);

createDefaultMemIntf Function

File

gfxu_image_utils.h

C

```
void createDefaultMemIntf(GFXU_MemoryIntf* memIntf);
```

Description

internal use only

GFX_ColorBilerp Function

Calculates bilinear interpolation between four colors

File

gfx_color.h

C

```
LIB_EXPORT GFX_Color GFX_ColorBilerp(GFX_Color c00, GFX_Color c01, GFX_Color c10, GFX_Color c11, uint32_t
xper, uint32_t yper, GFX_ColorMode mode);
```

Returns

GFX_Color - the result color

Parameters

Parameters	Description
GFX_Color c00	top left color input
GFX_Color c01	top right color input

GFX_Color c10	bottom left color input
GFX_Color c11	bottom right color input
uint32_t xper	percentage of interpolation in x [0-100]
uint32_t yper	percentage of interpolation in y [0-100]
GFX_ColorMode	input color mode

Function

GFX_Color GFX_ColorBilerp(GFX_Color c00,
GFX_Color c01,
GFX_Color c10,
GFX_Color c11,
uint32_t xper,
uint32_t yper,
GFX_ColorMode mode)

GFX_ColorBlend_RGBA_8888 Function

Blends two RGBA8888 colors together using their alpha channel values.

File

gfx_color.h

C

```
LIB_EXPORT GFX_Color GFX_ColorBlend_RGBA_8888(GFX_Color fore, GFX_Color back);
```

Returns

GFX_Color - the blended result color

Parameters

Parameters	Description
GFX_Color	the foreground color the background color

Function

GFX_Color GFX_ColorBlend_RGBA_8888(GFX_Color fore, GFX_Color back)

GFX_ColorChannelBlue Function

Used for getting the blue color channel of a given color value.

File

gfx_color.h

C

```
LIB_EXPORT uint32_t GFX_ColorChannelBlue(GFX_Color clr, GFX_ColorMode mode);
```

Returns

uint32_t - the blue color channel

Parameters

Parameters	Description
GFX_Color	the source color value
GFX_ColorMode	the source color mode

Function

uint32_t GFX_ColorChannelBlue(GFX_Color clr, GFX_ColorMode mode)

GFX_DivideRounding Function

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_DivideRounding(int32_t num, int32_t denom);
```

Description

This is function GFX_DivideRounding.

GFX_DrawBlit Function

Blits a buffer of pixels into the frame buffer.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawBlit(GFX_PixelBuffer* buffer, int32_t src_x, int32_t src_y, int32_t
src_width, int32_t src_height, int32_t dest_x, int32_t dest_y);
```

Returns

GFX_Result - Returns GFX_TRUE if the blit was drawn successfully. Otherwise returns GFX_FALSE.

Description

A pixel buffer is an array of pixel data that can be applied in bulk, or 'blit'ed, to the frame buffer. Pixel buffers may be of a different color mode and will be converted to match the destination frame buffer before application.

Parameters

Parameters	Description
buffer	the pointer to the source pixel buffer
src_x	the x component of the rectangle of the buffer to blit, usually 0
src_y	the y component of the rectangle of the buffer to blit, usually 0
src_width	width of the rectangle of the buffer to blit, usually the entire with of the source buffer
src_height	height of the rectangle of the buffer to blit, usually the entire height of the source buffer
dest_x	the x position to blit the source rectangle in the destination buffer the y position to blit the source rectangle in the destination buffer

Function

GFX_Result GFX_Result GFX_DrawBlit(GFX_PixelBuffer* buffer,
int32_t src_x,
int32_t src_y,
int32_t src_width,
int32_t src_height,
int32_t dest_x,
int32_t dest_y);

GFX_DrawCircle Function

Draws a circle from using the specified dimensions and the current draw state.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawCircle(int32_t x, int32_t y, int32_t radius);
```

Returns

GFX_Result - Returns GFX_TRUE if the circle was drawn successfully. Otherwise returns GFX_FALSE.

Parameters

Parameters	Description
x	the x component of the origin position
y	the y component of the origin position
radius	the radius of the circle in pixels

Function

GFX_Result GFX_Result GFX_DrawCircle(int32_t x,
int32_t y,
int32_t radius);

GFX_DrawLine Function

Draws a line from (x1,y1) to (x2,y2) using the current draw state.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawLine(int32_t x1, int32_t y1, int32_t x2, int32_t y2);
```

Returns

GFX_Result - Returns GFX_TRUE if the line was drawn successfully. Otherwise returns GFX_FALSE.

Parameters

Parameters	Description
x1	the x component of the first coordinate of the line
y1	the y component of the first coordinate of the line
x2	the x component of the second coordinate of the line
y2	the y component of the second coordinate of the line

Function

GFX_Result GFX_Result GFX_DrawLine(int32_t x1,
int32_t y1,
int32_t x2,
int32_t y2);

GFX_DrawPixel Function

Sets the pixel at X and Y using the current draw state.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawPixel(int32_t x, int32_t y);
```

Returns

GFX_Result - Returns GFX_TRUE if the pixel was drawn successfully. Otherwise returns GFX_FALSE.

Parameters

Parameters	Description
x	the x coordinate of the pixel
y	the y coordinate of the pixel

Function

GFX_Result GFX_DrawPixel(int32_t x, int32_t y);

GFX_DrawRect Function

Draws a rectangle using the specified dimensions and the current draw state.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawRect(int32_t x, int32_t y, int32_t width, int32_t height);
```

Returns

GFX_Result - Returns GFX_TRUE if the rectangle was drawn successfully. Otherwise returns GFX_FALSE.

Description

Draws a rectangle using the coordinates: x,y x + width - 1, y

x, y + height - 1 x + width - 1, y + height - 1

Parameters

Parameters	Description
x	the x position of the top left point of the rectangle
y	the y position of the top left point of the rectangle
width	the width of the rectangle in pixels
height	the height of the rectangle in pixels

Function

GFX_Result GFX_Result GFX_DrawLine(int32_t x,

int32_t x,

int32_t width,

int32_t height);

GFX_DrawStretchBlit Function

Blits a buffer of pixels into the frame buffer.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawStretchBlit(GFX_PixelBuffer* buffer, int32_t src_x, int32_t src_y, int32_t
src_width, int32_t src_height, int32_t dest_x, int32_t dest_y, int32_t dest_width, int32_t dest_height);
```

Returns

GFX_Result - Returns GFX_TRUE if the blit was drawn successfully. Otherwise returns GFX_FALSE.

Description

A pixel buffer is an array of pixel data that can be applied in bulk, or 'blit'ed, to the frame buffer. Pixel buffers may be of a different color mode and will be converted to match the destination frame buffer before application. This version can resize the source data before blitting. The option GFXF_RESIZE_METHOD selects the resize technique.

Parameters

Parameters	Description
buffer	the pointer to the source pixel buffer
src_x	the x component of the rectangle of the buffer to blit, usually 0
src_y	the y component of the rectangle of the buffer to blit, usually 0
src_width	width of the rectangle of the buffer to blit, usually the entire with of the source buffer
src_height	height of the rectangle of the buffer to blit, usually the entire height of the source buffer

dest_x	the x position to blit the source rectangle in the destination buffer the y position to blit the source rectangle in the destination buffer
dest_width	the desired resize width
dest_height	the desired resize height

Function

GFX_Result GFX_DrawStretchBlit(GFX_PixelBuffer* buffer,
int32_t src_x,
int32_t src_y,
int32_t src_width,
int32_t src_height,
int32_t dest_x,
int32_t dest_y,
int32_t dest_width,
int32_t dest_height);

GFX_LayerFromOrientedSpace Function

Transforms a layer oriented space to screen space.

File

gfx_layer.h

C

```
void GFX_LayerFromOrientedSpace(GFX_Rect* displayRect, GFX_Layer* layer, GFX_Orientation ori, GFX_Bool
mirrored);
```

Returns

void

Parameters

Parameters	Description
GFX_Rect* dispayRect	the rectangle of the display
GFX_Layert* layer	the layer
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

void GFX_LayerFromOrientedSpace(GFX_Rect* displayRect,
GFX_Layer* layer,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_LayerPointFromOrientedSpace Function

Transforms a point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.

File

gfx_layer.h

C

```
LIB_EXPORT GFX_Point GFX_LayerPointFromOrientedSpace(const GFX_Layer* layer, const GFX_Point* point,
GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Layer* layer	the layer
const GFX_Point* point	the point
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Point GFX_LayerPointFromOrientedSpace(const GFX_Layer* layer,
const GFX_Point* point,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_LayerPointToOrientedSpace Function

Transforms a point from screen space to oriented space given a layer, a display orientation, and a mirroring setting.

File

gfx_layer.h

C

```
LIB_EXPORT GFX_Point GFX_LayerPointToOrientedSpace(const GFX_Layer* layer, const GFX_Point* point,
GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Layer* layer	the layer
const GFX_Point* point	the point to transform
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Point GFX_LayerPointToOrientedSpace(const GFX_Rect* layerRect,
const GFX_Point* point,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_LayerRectFromOrientedSpace Function

Transforms a layer point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.

File

gfx_layer.h

C

```
LIB_EXPORT GFX_Rect GFX_LayerRectFromOrientedSpace(const GFX_Layer* layer, const GFX_Rect* rect,
GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Layer* layer	the layer
const GFX_Rect* rect	the rectangle to transform

GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Rect GFX_LayerRectFromOrientedSpace(const GFX_Layer* layer,
const GFX_Rect* rect,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_LayerRectToOrientedSpace Function

Transforms a rectangle from screen space to oriented space given a layer, a display orientation, and a mirroring setting.

File

gfx_layer.h

C

```
LIB_EXPORT GFX_Rect GFX_LayerRectToOrientedSpace(const GFX_Layer* layer, const GFX_Rect* rect,
GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Layer* layer	the layer
const GFX_Rect* rect	the rectangle to transform
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Rect GFX_LayeRectToOrientedSpace(const GFX_Rect* layerRect,
const GFX_Rect* rect,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_LayerToOrientedSpace Function

Transforms a layer from screen space to oriented space.

File

gfx_layer.h

C

```
void GFX_LayerToOrientedSpace(GFX_Rect* displayRect, GFX_Layer* layer, GFX_Orientation ori, GFX_Bool
mirrored);
```

Returns

void

Parameters

Parameters	Description
GFX_Rect* dispayRect	the rectangle of the display
GFX_Rect* layer	the layer
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

void GFX_LayerToOrientedSpace(GFX_Rect* displayRect,

GFX_Layer* layer,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_PercentOfDec Function

File

gfx_math.h

C

```
LIB_EXPORT void GFX_PercentOfDec(uint32_t num, uint32_t percent, uint32_t* whl, uint32_t* dec);
```

Description

This is function GFX_PercentOfDec.

GFX_RectClipAdj Function

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Rect GFX_RectClipAdj(const GFX_Rect* l_rect, const GFX_Rect* r_rect, GFX_Rect* adj);
```

Description

This is function GFX_RectClipAdj.

GFX_RectFromPoints Function

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Rect GFX_RectFromPoints(const GFX_Point* p1, const GFX_Point* p2);
```

Description

This is function GFX_RectFromPoints.

GFX_RectSplit Function

File

gfx_rect.h

C

```
LIB_EXPORT uint32_t GFX_RectSplit(const GFX_Rect* sub, const GFX_Rect* obj, GFX_Rect res[4]);
```

Description

This is function GFX_RectSplit.

GFX_RectToPoints Function

File

gfx_rect.h

C

```
LIB_EXPORT void GFX_RectToPoints(const GFX_Rect* rect, GFX_Point* p1, GFX_Point* p2);
```

Description

This is function GFX_RectToPoints.

GFX_UtilMirrorPoint Function

Reorients a point to a given mirrored orientation.

File

gfx_util.h

C

```
GFX_Point GFX_UtilMirrorPoint(const GFX_Point* point, const GFX_Rect* rect, GFX_Orientation ori);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Point* point	the point to reorient
const GFX_Rect* rect	the bounding rectangle space
GFX_Orientation	the orientation setting

Function

void GFX_UtilMirrorPoint(const GFX_Point* point,
const GFX_Rect* rect,
GFX_Orientation ori)

GFX_UtilOrientPoint Function

Reorients a point to a given orthogonal orientation.

File

gfx_util.h

C

```
GFX_Point GFX_UtilOrientPoint(const GFX_Point* point, const GFX_Rect* rect, GFX_Orientation ori);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Point* point	the point to reorient
const GFX_Rect* rect	the bounding rectangle space
GFX_Orientation	the orientation setting

Function

void GFX_UtilOrientPoint(const GFX_Point* point,
const GFX_Rect* rect,
GFX_Orientation ori)

GFX_UtilPointFromOrientedSpace Function

Transforms a point from an oriented rectangle space to an outer space given a display orientation and a mirroring setting.

File

gfx_util.h

C

```
GFX_Point GFX_UtilPointFromOrientedSpace(const GFX_Rect* outerRect, const GFX_Rect* innerRect, const
GFX_Point* pnt, GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Rect* outerRect	the outer rectangle
const GFX_Rect* subRect	the inner rectangle
const GFX_Point* point	the point
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Point GFX_UtilPointFromOrientedSpace(const GFX_Rect* displayRect,
const GFX_Rect* innerRect,
const GFX_Point* pnt,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_UtilPointToOrientedSpace Function

Transforms a point to an oriented rectangle space to an outer space given a display orientation and a mirroring setting.

File

gfx_util.h

C

```
GFX_Point GFX_UtilPointToOrientedSpace(const GFX_Rect* outerRect, const GFX_Rect* innerRect, const
GFX_Point* pnt, GFX_Orientation ori, GFX_Bool mirrored);
```

Returns

GFX_Point

Parameters

Parameters	Description
const GFX_Rect* outerRect	the outer rectangle
const GFX_Rect* subRect	the inner rectangle
const GFX_Point* point	the point
GFX_Orientation	the orientation setting
GFX_Bool	the mirroring setting

Function

GFX_Point GFX_UtilPointToOrientedSpace(const GFX_Rect* displayRect,
const GFX_Rect* layerRect,
const GFX_Point* point,
GFX_Orientation ori,
GFX_Bool mirrored)

GFX_UtilSizeFromOrientedSpace Function

Transforms a size tuple from oriented space to screen space

File

gfx_util.h

C

```
GFX_Size GFX_UtilSizeFromOrientedSpace(const GFX_Size* size, GFX_Orientation ori);
```

Returns

GFX_Size

Parameters

Parameters	Description
const GFX_Size* size	the size dimension
GFX_Orientation	the orientation setting

Function

GFX_Size GFX_UtilSizeFromOrientedSpace(const GFX_Size* size, GFX_Orientation ori)

GFX_UtilSizeToOrientedSpace Function

Transforms a size tuple from screen space to oriented space

File

gfx_util.h

C

```
GFX_Size GFX_UtilSizeToOrientedSpace(const GFX_Size* size, GFX_Orientation ori);
```

Returns

GFX_Size

Parameters

Parameters	Description
const GFX_Size* size	the size dimension
GFX_Orientation	the orientation setting

Function

void GFX_UtilSizeToOrientedSpace(const GFX_Size* size, GFX_Orientation ori)

GFX_UtilSortPointsX Function

Sorts two points in the X axis

File

gfx_util.h

C

```
void GFX_UtilSortPointsX(GFX_Point* p1, GFX_Point* p2);
```

Returns

void

Parameters

Parameters	Description
GFX_Point* p1	the first point to sort
GFX_Point* p2	the second point to sort

Function

void GFX_UtilSortPointsX(GFX_Point* p1, GFX_Point* p2)

GFX_UtilSortPointsY Function

Sorts two points in the Y axis

File

gfx_util.h

C

```
void GFX_UtilSortPointsY(GFX_Point* p1, GFX_Point* p2);
```

Returns

void

Parameters

Parameters	Description
GFX_Point* p1	the first point to sort
GFX_Point* p2	the second point to sort

Function

void GFX_UtilSortPointsY(GFX_Point* p1, GFX_Point* p2)

GFX_RectCombine Function

Combines the area of two rectangles into a single rectangle.

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Rect GFX_RectCombine(const GFX_Rect* l_rect, const GFX_Rect* r_rect);
```

Returns

void

Parameters

Parameters	Description
const GFX_Rect* l_rect	the first rectangle
const GFX_Rect* r_rect	the second rectangle

Function

GFX_Rect GFX_RectCombine(const GFX_Rect* l_rect,
const GFX_Rect* r_rect)

GFX_DrawDirectBlit Function

Blits a buffer of pixels into the frame buffer without performing per-pixel operations on the data.

File

gfx_draw.h

C

```
LIB_EXPORT GFX_Result GFX_DrawDirectBlit(GFX_PixelBuffer* buffer, int32_t src_x, int32_t src_y, int32_t
src_width, int32_t src_height, int32_t dest_x, int32_t dest_y);
```

Returns

GFX_Result - Returns GFX_TRUE if the blit was drawn successfully. Otherwise returns GFX_FALSE.

Description

A pixel buffer is an array of pixel data that can be applied in bulk, or 'blit'ed, to the frame buffer. This method will not perform per-pixel operations on the incoming data. The incoming pixel data color format must match the format of the current target buffer. Area clipping operations are still performed if enabled.

Parameters

Parameters	Description
buffer	the pointer to the source pixel buffer
src_x	the x component of the rectangle of the buffer to blit, usually 0
src_y	the y component of the rectangle of the buffer to blit, usually 0

src_width	width of the rectangle of the buffer to blit, usually the entire with of the source buffer
src_height	height of the rectangle of the buffer to blit, usually the entire height of the source buffer
dest_x	the x position to blit the source rectangle in the destination buffer the y position to blit the source rectangle in the destination buffer

Function

GFX_Result GFX_DrawDirectBlit(GFX_PixelBuffer* buffer,
int32_t src_x,
int32_t src_y,
int32_t src_width,
int32_t src_height,
int32_t dest_x,
int32_t dest_y);

GFX_RectCompare Function

File

gfx_rect.h

C

```
LIB_EXPORT int32_t GFX_RectCompare(const GFX_Rect* l, const GFX_Rect* r);
```

Description

This is function GFX_RectCompare.

GFX_RectsAreSimilar Function

File

gfx_rect.h

C

```
LIB_EXPORT GFX_Bool GFX_RectsAreSimilar(const GFX_Rect* l, const GFX_Rect* r);
```

Description

This is function GFX_RectsAreSimilar.

GFX_ScaleIntegerSigned Function

Scales a signed integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.

File

gfx_math.h

C

```
LIB_EXPORT int32_t GFX_ScaleIntegerSigned(int32_t num, int32_t oldMax, int32_t newMax);
```

Returns

int32_t - the number as defined in the new number range

Parameters

Parameters	Description
num	the number to consider
oldMax	the old range maximum
newMax	the new range maximum

Function

int32_t GFX_ScaleIntegerSigned(int32_t num, int32_t oldMax, int32_t newMax);

b) Data Types and Constants

GFX_ColorMode_t Enumeration

List of available color modes.

File

gfx_color.h

C

```
enum GFX_ColorMode_t {
  GFX_COLOR_MODE_GS_8 = 0x0,
  GFX_COLOR_MODE_RGB_332,
  GFX_COLOR_MODE_RGB_565,
  GFX_COLOR_MODE_RGBA_5551,
  GFX_COLOR_MODE_RGB_888,
  GFX_COLOR_MODE_RGBA_8888,
  GFX_COLOR_MODE_ARGB_8888,
  GFX_COLOR_MODE_YUV,
  GFX_COLOR_MODE_INDEX_1,
  GFX_COLOR_MODE_INDEX_4,
  GFX_COLOR_MODE_INDEX_8,
  GFX_COLOR_MODE_LAST = GFX_COLOR_MODE_INDEX_8
};
```

Description

Enumeration: GFX_ColorMode_t

GFX_DriverInfo_t Structure

A driver description structure.

File

gfx_driver_interface.h

C

```
struct GFX_DriverInfo_t {
  char name[16];
  GFX_ColorMask color_formats;
  uint32_t layer_count;
};
```

Description

Structure: GFX_DriverInfo_t

name - a short human-readable name. color formats - a mask of supported color formats layer_count - number of layers supported by the driver

Remarks

None.

GFX_Point_t Structure

A two dimensional Cartesian point.

File

gfx_common.h

C

```
struct GFX_Point_t {
  int32_t x;
  int32_t y;
};
```

Description

Structure: GFX_Point_t

GFX_Rect_t Structure

A rectangle definition.

File

gfx_common.h

C

```
struct GFX_Rect_t {
  int32_t x;
  int32_t y;
  int32_t width;
  int32_t height;
};
```

Description

Structure: GFX_Rect_t

begin_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* begin_FnPtr)(void);
```

Description

This is type begin_FnPtr.

boolGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Bool (* boolGet_FnPtr)(void);
```

Description

This is type boolGet_FnPtr.

boolSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* boolSet_FnPtr)(GFX_Bool);
```

Description

This is type boolSet_FnPtr.

brightnessGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* brightnessGet_FnPtr)(void);
```

Description

This is type brightnessGet_FnPtr.

brightnessRangeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* brightnessRangeGet_FnPtr)(uint32_t*, uint32_t*);
```

Description

This is type brightnessRangeGet_FnPtr.

brightnessSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* brightnessSet_FnPtr)(uint32_t);
```

Description

This is type brightnessSet_FnPtr.

colorModeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_ColorMode (* colorModeGet_FnPtr)(void);
```

Description

This is type colorModeGet_FnPtr.

colorModeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* colorModeSet_FnPtr)(GFX_ColorMode);
```

Description

This is type colorModeSet_FnPtr.

destroy_FnPtr Type

File

gfx_hal.h

C

```
typedef void (* destroy_FnPtr)(GFX_Context*);
```

Description

This is type destroy_FnPtr.

drawAlphaValueGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* drawAlphaValueGet_FnPtr)(void);
```

Description

This is type drawAlphaValueGet_FnPtr.

drawAlphaValueSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawAlphaValueSet_FnPtr)(uint32_t);
```

Description

This is type drawAlphaValueSet_FnPtr.

drawBlendModeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_BlendMode (* drawBlendModeGet_FnPtr)(void);
```

Description

This is type drawBlendModeGet_FnPtr.

drawBlendModeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawBlendModeSet_FnPtr)(GFX_BlendMode);
```

Description

This is type drawBlendModeSet_FnPtr.

drawBlit_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawBlit_FnPtr)(const GFX_PixelBuffer*, const GFX_Rect*, const GFX_Point*, const
GFX_DrawState*);
```

Description

This is type drawBlit_FnPtr.

drawCircle_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawCircle_FnPtr)(const GFX_Point*, int32_t, const GFX_DrawState*);
```

Description

This is type drawCircle_FnPtr.

drawClipRectGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawClipRectGet_FnPtr)(GFX_Rect*);
```

Description

This is type drawClipRectGet_FnPtr.

drawClipRectSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawClipRectSet_FnPtr)(const GFX_Rect*);
```

Description

This is type drawClipRectSet_FnPtr.

drawColorGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Color (* drawColorGet_FnPtr)(void);
```

Description

This is type drawColorGet_FnPtr.

drawColorSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawColorSet_FnPtr)(GFX_Color);
```

Description

This is type drawColorSet_FnPtr.

drawGradientColorGet_FnPtr Type

File

gfx_hal.h

C

```
typedef void (* drawGradientColorGet_FnPtr)(GFX_Color*, GFX_Color*, GFX_Color*, GFX_Color*);
```

Description

This is type drawGradientColorGet_FnPtr.

drawGradientColorSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawGradientColorSet_FnPtr)(GFX_Color, GFX_Color, GFX_Color, GFX_Color);
```

Description

This is type drawGradientColorSet_FnPtr.

drawLine_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawLine_FnPtr)(const GFX_Point*, const GFX_Point*, const GFX_DrawState*);
```

Description

This is type drawLine_FnPtr.

drawLock_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawLock_FnPtr)(void);
```

Description

This is type drawLock_FnPtr.

drawMaskValueGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* drawMaskValueGet_FnPtr)(void);
```

Description

This is type drawMaskValueGet_FnPtr.

drawMaskValueSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawMaskValueSet_FnPtr)(uint32_t);
```

Description

This is type drawMaskValueSet_FnPtr.

drawModeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_DrawMode (* drawModeGet_FnPtr)(void);
```

Description

This is type drawModeGet_FnPtr.

drawModeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawModeSet_FnPtr)(GFX_DrawMode);
```

Description

This is type drawModeSet_FnPtr.

drawPaletteGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawPaletteGet_FnPtr)(GFX_PixelBuffer*);
```

Description

This is type drawPaletteGet_FnPtr.

drawPaletteSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawPaletteSet_FnPtr)(const GFX_PixelBuffer*);
```

Description

This is type drawPaletteSet_FnPtr.

drawPixel_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawPixel_FnPtr)(const GFX_Point*, const GFX_DrawState*);
```

Description

This is type drawPixel_FnPtr.

drawRect_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawRect_FnPtr)(const GFX_Rect*, const GFX_DrawState*);
```

Description

This is type drawRect_FnPtr.

drawThicknessGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* drawThicknessGet_FnPtr)(void);
```

Description

This is type drawThicknessGet_FnPtr.

drawThicknessSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawThicknessSet_FnPtr)(uint32_t);
```

Description

This is type drawThicknessSet_FnPtr.

drawUnlock_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawUnlock_FnPtr)(void);
```

Description

This is type drawUnlock_FnPtr.

GFX_AntialiasMode Enumeration

Enables anti-aliased drawing hint

File

gfx_draw.h

C

```
typedef enum GFX_AntialiasMode_t {
  GFX_ANTIALIAS_OFF = 0x0,
  GFX_ANTIALIAS_ON = 0x1
} GFX_AntialiasMode;
```

Description

Enumeration: GFX_AntialiasMode_t

Remarks

None.

GFX_BitsPerPixel Enumeration

List of available bits-per-pixel sizes.

File

gfx_color.h

C

```
typedef enum GFX_BitsPerPixel_t {
  GFX_BPP1,
  GFX_BPP4,
  GFX_BPP8,
  GFX_BPP16,
  GFX_BPP24,
  GFX_BPP32
} GFX_BitsPerPixel;
```

Description

Enumeration: GFX_BitsPerPixel_t

GFX_BlendMode Enumeration

Blend mode masks

File

gfx_common.h

C

```
typedef enum GFX_BlendMode_t {
  GFX_BLEND_NONE = 0x0,
  GFX_BLEND_CHANNEL = 0x1,
  GFX_BLEND_GLOBAL = 0x2,
  GFX_BLEND_ALL = GFX_BLEND_CHANNEL|GFX_BLEND_GLOBAL
} GFX_BlendMode;
```

Description

Enumeration: GFX_BlendMode_t

GFX_Bool Type

File

gfx_common.h

C

```
typedef uint32_t GFX_Bool;
```

Description

This is type GFX_Bool.

GFX_Buffer Type

File

gfx_common.h

C

```
typedef void* GFX_Buffer;
```

Description

This is type GFX_Buffer.

GFX_BufferSelection Enumeration

Buffer selector used when querying layers for certain buffer states.

File

gfx_common.h

C

```
typedef enum GFX_BufferSelection_t {
  GFX_BUFFER_READ,
  GFX_BUFFER_WRITE
} GFX_BufferSelection;
```

Description

Enumeration: GFX_BufferSelection_t

GFX_BufferState Enumeration

Frame buffer memory states

File

gfx_common.h

C

```
typedef enum GFX_BufferState_t {
  GFX_BS_NONE = 0x0,
  GFX_BS_ADDRESS,
  GFX_BS_MALLOC,
  GFX_BS_MANAGED
} GFX_BufferState;
```

Description

Enumeration: GFX_BufferState_t

address - The buffer is set to a discrete address. This could be an address located in DDR memory, a buffer allocated by the application, or some other location.

malloc - The buffer has been dynamically allocated from some form of heap or memory manager. These can be freed as desired.

managed - The buffer is owned by the system and cannot be allocated or freed, it just is. This is common in systems where the memory is owned by the graphics driver or the memory resides on the graphics controller.

Remarks

None.

GFX_Calloc_FnPtr Type

Simple wrapper around the standard calloc function pointer. Used for redirecting memory allocation to other memory management systems.

File

gfx_common.h

C

```
typedef void* (* GFX_Calloc_FnPtr)(size_t, size_t);
```

Description

Function pointer

Function

typedef void* (*GFX_Calloc_FnPtr)(size_t, size_t);

GFX_Color Type

File

gfx_common.h

C

```
typedef uint32_t GFX_Color;
```

Description

This is type GFX_Color.

GFX_ColorMask Enumeration

Maskable list of color valies.

File

gfx_color.h

C

```
typedef enum GFX_ColorMask_t {
  GFX_COLOR_MASK_GS_8 = 0x1,
  GFX_COLOR_MASK_RGB_332 = 0x4,
  GFX_COLOR_MASK_RGB_565 = 0x8,
  GFX_COLOR_MASK_RGBA_5551 = 0x10,
  GFX_COLOR_MASK_RGB_888 = 0x20,
  GFX_COLOR_MASK_RGBA_8888 = 0x40,
  GFX_COLOR_MASK_ARGB_8888 = 0x80,
  GFX_COLOR_MASK_YUV = 0x100,
  GFX_COLOR_MASK_ALL =
GFX_COLOR_MASK_GS_8|GFX_COLOR_MASK_RGB_332|GFX_COLOR_MASK_RGB_565|GFX_COLOR_MASK_RGBA_5551|GFX_COLOR_MASK_RG
B_888|GFX_COLOR_MASK_RGBA_8888|GFX_COLOR_MASK_ARGB_8888|GFX_COLOR_MASK_YUV
} GFX_ColorMask;
```

Description

Enumeration: GFX_ColorMask_t

GFX_ColorMode Type

List of available color modes.

File

gfx_common.h

C

```
typedef enum GFX_ColorMode_t GFX_ColorMode;
```

Description

Enumeration: GFX_ColorMode_t

GFX_ColorModeInfo Structure

Struct that provides information about a color mode.

File

gfx_color.h

C

```
typedef struct GFX_ColorModeInfo_t {
  uint32_t size;
  uint32_t bpp;
  GFX_BitsPerPixel bppOrdinal;
  struct masks {
    uint32_t red;
    uint32_t green;
    uint32_t blue;
    uint32_t alpha;
  } mask;
  struct shifts {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
    uint8_t alpha;
  } shift;
} GFX_ColorModeInfo;
```

Description

Structure: GFX_ColorModeInfo_t

size - size in bytes bpp - bpp value bppOrdinal - bpp enum value masks - the masks used for extracting individual color channel information

Remarks

None.

GFX_ColorName Enumeration

Color name reference table

File

gfx_color.h

C

```
typedef enum GFX_ColorName_t {
  GFX_COLOR_BLACK,
  GFX_COLOR_WHITE,
  GFX_COLOR_RED,
  GFX_COLOR_LIME,
  GFX_COLOR_BLUE,
  GFX_COLOR_YELLOW,
  GFX_COLOR_CYAN,
  GFX_COLOR_MAGENTA,
  GFX_COLOR_SILVER,
  GFX_COLOR_DARKGRAY,
  GFX_COLOR_GRAY,
  GFX_COLOR_LIGHTGRAY,
  GFX_COLOR_MAROON,
  GFX_COLOR_OLIVE,
  GFX_COLOR_GREEN,
  GFX_COLOR_PURPLE,
  GFX_COLOR_TEAL,
  GFX_COLOR_NAVY,
  GFX_COLOR_LAST
} GFX_ColorName;
```

Description

Structure: GFX_ColorName_t

GFX_Context Structure

An instance of the hardware abstraction layer.

File

gfx_context.h

C

```
typedef struct GFX_Context_t {
  GFX_Display display_idx;
  GFX_DisplayInfo* display_info;
  struct {
    uint32_t count;
    uint32_t active_idx;
    GFX_Layer* active;
    GFX_Layer* layers;
  } layer;
  uint32_t brightness;
  GFX_Orientation orientation;
  GFX_Bool mirrored;
  GFX_Bool layerSwapSync;
  GFX_ColorMode colorMode;
  GFX_GlobalPalette globalPalette;
  GFX_DrawState draw;
  GFX_SyncCallback_FnPtr vsyncCB;
  GFX_SyncCallback_FnPtr hsyncCB;
  GFX_HAL hal;
  GFX_MemoryIntf memory;
  void* driver_data;
} GFX_Context;
```

Members

Members	Description
GFX_SyncCallback_FnPtr vsyncCB;	GFX_DRAW_PIPELINE_ENABLED

Description

Stucture: GFX_Context_t

The context is an instance of the hardware abstraction layer. It is essentially the marriage between a graphics driver, a display description, and possibly a graphics processor. It contains the data that describes the layout of the display, the buffers that store the display data, the current draw state, and the function map that controls everything.

Members: display_idx - the display associated with this context display_info - a pointer to the information for the display for this context

layer.count - the number of existing layers layer.active_idx - the index of the active layer layer.active - the pointer to the active layer layer.layers - the array of layers for this context

brightness - the brightness setting for this context orientation - the orientation mode for this context mirrored - the mirror mode for this context

colorMode - the color mode for this context, all buffers of all layers use this mode

draw - the current draw state of this context

vsyncCB - the callback to invoke when the driver enters vsync mode hsyncCB - the callback to invoke when the driver enters hsync mode

hal - the function table for this context

memory - the memory management interface for this context

driver_data - a pointer that can be used for driver-specific data purposes

Remarks

None.

GFX_Display Type

File

gfx_common.h

C

```
typedef uint32_t GFX_Display;
```

Description

This is type GFX_Display.

GFX_DisplayInfo Structure

Describes a graphical display device.

File

gfx_display.h

C

```
typedef struct GFX_DisplayInfo_t {
  const char name[16];
  GFX_ColorMask color_formats;
  GFX_Rect rect;
  struct attributes_t {
    int8_t data_width;
    struct horizontal_t {
      int8_t pulse_width;
      int8_t back_porch;
      int8_t front_porch;
    } horz;
    struct vertical_t {
      int8_t pulse_width;
      int8_t back_porch;
      int8_t front_porch;
    } vert;
    int32_t inv_left_shift;
  } attributes;
} GFX_DisplayInfo;
```

Description

Structure: GFX_DisplayInfo_t

name - a short human-readable name color_formats - mask of color formats this display supports rect - the size of the display

Remarks

None.

GFX_DrawMode Enumeration

Gradient draw modes.

File

gfx_draw.h

C

```
typedef enum GFX_DrawMode_t {
  GFX_DRAW_LINE = 0x0,
  GFX_DRAW_FILL,
  GFX_DRAW_GRADIENT_LEFT_RIGHT,
  GFX_DRAW_GRADIENT_TOP_BOTTOM
} GFX_DrawMode;
```

Description

Enumeration: GFX_DrawMode_t

line - draws the outline of a shape fill - draws a filled shape gradient left/right - draws a gradient from left to right, uses the first two gradient colors gradient top/bottom - draws a gradient from top to bottom, uses the first two gradient colors

Remarks

None.

GFX_DrawState Structure

A list of drawing hints for shape drawing algorithms

File

gfx_draw.h

C

```
typedef struct GFX_DrawState_t {
  GFX_DrawMode mode;
  GFX_Color color;
  GFX_ColorMode colorMode;
```

```
    struct {
      GFX_Color c0;
      GFX_Color c1;
      GFX_Color c2;
      GFX_Color c3;
    } gradient;
    GFX_PixelBuffer palette;
    const GFX_PixelBuffer* target;
    GFX_Rect targetClipRect;
    GFX_BlendMode blendMode;
    GFX_Bool alphaEnable;
    uint32_t globalAlphaValue;
    GFX_Bool maskEnable;
    uint32_t maskValue;
    GFX_Bool antialias;
    uint32_t thickness;
    GFX_Bool clipEnable;
    GFX_Rect clipRect;
    GFX_ResizeMode resizeMode;
    GFX_PipelineMode pipelineMode;
    GFX_DrawPipeline* pipeline;
  } GFX_DrawState;
```

Description

Structure: GFX_DrawState_t

mode - the shape drawing mode

color - the draw color

gradient - the list of gradient colors

palette - the palette lookup table for blits

alphaEnable - indicates if alpha blending is enabled alphaValue - the desired alpha blending amount

maskEnable - indicates if pixel masking is enabled maskValue - the mask/transparency color value

clipEnable - indicate of pixel clipping is enabled clipRect - the pixel clipping rectangle

Remarks

None.

GFX_Driver Type

File

gfx_common.h

C

```
typedef uint32_t GFX_Driver;
```

Description

This is type GFX_Driver.

GFX_DriverInfo Type

A driver description structure.

File

gfx_hal.h

C

```
typedef struct GFX_DriverInfo_t GFX_DriverInfo;
```

Description

Structure: GFX_DriverInfo_t

name - a short human-readable name. color formats - a mask of supported color formats layer_count - number of layers supported by the driver

Remarks

None.

GFX_Flag Enumeration

Hardware abstraction state interface flags. See gfx.h for a comprehensive description.

File

gfx_common.h

C

```
typedef enum GFX_Flag_t {
  GFXF_NONE = 0,
  GFXF_DISPLAY_COUNT,
  GFXF_DISPLAY_INFO,
  GFXF_DRIVER_COUNT,
  GFXF_DRIVER_INFO,
  GFXF_BRIGHTNESS_RANGE,
  GFXF_BRIGHTNESS,
  GFXF_VSYNC_CALLBACK,
  GFXF_HSYNC_CALLBACK,
  GFXF_ORIENTATION,
  GFXF_MIRRORED,
  GFXF_COLOR_MODE,
  GFXF_GLOBAL_PALETTE,
  GFXF_LAYER_COUNT,
  GFXF_LAYER_ACTIVE,
  GFXF_LAYER_ENABLED,
  GFXF_LAYER_VISIBLE,
  GFXF_LAYER_VSYNC,
  GFXF_LAYER_INVALID,
  GFXF_LAYER_SWAP_SYNC,
  GFXF_LAYER_SWAP,
  GFXF_LAYER_POSITION,
  GFXF_LAYER_SIZE,
  GFXF_LAYER_ALPHA_ENABLE,
  GFXF_LAYER_ALPHA_AMOUNT,
  GFXF_LAYER_MASK_ENABLE,
  GFXF_LAYER_MASK_COLOR,
  GFXF_LAYER_BUFFER_COUNT,
  GFXF_LAYER_BUFFER_ADDRESS,
  GFXF_LAYER_BUFFER_COHERENT,
  GFXF_LAYER_BUFFER_ALLOCATE,
  GFXF_LAYER_BUFFER_FREE,
  GFXF_DRAW_PIPELINE_MODE,
  GFXF_DRAW_MODE,
  GFXF_DRAW_COLOR,
  GFXF_DRAW_GRADIENT_COLOR,
  GFXF_DRAW_PALETTE,
  GFXF_DRAW_TARGET,
  GFXF_DRAW_THICKNESS,
  GFXF_DRAW_BLEND_MODE,
  GFXF_DRAW_RESIZE_MODE,
  GFXF_DRAW_ALPHA_ENABLE,
  GFXF_DRAW_ALPHA_VALUE,
  GFXF_DRAW_MASK_ENABLE,
  GFXF_DRAW_MASK_VALUE,
  GFXF_DRAW_CLIP_ENABLE,
  GFXF_DRAW_CLIP_RECT,
  GFXF_LAST_FLAG
} GFX_Flag;
```

Description

Enumeration: GFX_Flag_t

GFX_FrameBuffer Structure

A frame buffer is a wrapper around a pixel buffer construct that is used by display drivers to manage frame buffers.

File

gfx_layer.h

C

```
typedef struct GFX_FrameBuffer_t {
  GFX_PixelBuffer pb;
  GFX_BufferState state;
  GFX_Bool coherent;
  void* driver_data;
} GFX_FrameBuffer;
```

Description

Structure: GFX_FrameBuffer_t

pb - The pixel buffer description of the frame buffer. state - The state of the frame buffer. coherent - Indicates if the frame buffer is allocated from coherent dynamic memory

Remarks

None.

GFX_Free_FnPtr Type

Simple wrapper around the standard free function pointer. Used for redirecting memory free to other memory management systems.

File

gfx_common.h

C

```
typedef void (* GFX_Free_FnPtr)(void*);
```

Description

Function pointer

Function

typedef void (*GFX_Free_FnPtr)(void*);

GFX_HAL Structure

File

gfx_hal.h

C

```
typedef struct GFX_HAL_t {
  initialize_FnPtr initialize;
  destroy_FnPtr destroy;
  begin_FnPtr begin;
  end_FnPtr end;
  update_FnPtr update;
  brightnessRangeGet_FnPtr brightnessRangeGet;
  brightnessGet_FnPtr brightnessGet;
  brightnessSet_FnPtr brightnessSet;
  syncCallbackGet_FnPtr vsyncCallbackGet;
  syncCallbackSet_FnPtr vsyncCallbackSet;
  syncCallbackGet_FnPtr hsyncCallbackGet;
  syncCallbackSet_FnPtr hsyncCallbackSet;
  orientationGet_FnPtr orientationGet;
  orientationSet_FnPtr orientationSet;
  boolGet_FnPtr mirroringGet;
  boolSet_FnPtr mirroringSet;
  colorModeGet_FnPtr colorModeGet;
  colorModeSet_FnPtr colorModeSet;
  globalPaletteGet_FnPtr globalPaletteGet;
  globalPaletteSet_FnPtr globalPaletteSet;
  layerActiveGet_FnPtr layerActiveGet;
  layerActiveSet_FnPtr layerActiveSet;
  boolGet_FnPtr layerEnabledGet;
  boolSet_FnPtr layerEnabledSet;
  layerBufferCountGet_FnPtr layerBufferCountGet;
  layerBufferCountSet_FnPtr layerBufferCountSet;
  layerBufferAddressGet_FnPtr layerBufferAddressGet;
```

```
    layerBufferAddressSet_FnPtr layerBufferAddressSet;
    layerBufferCoherentGet_FnPtr layerBufferCoherentGet;
    layerBufferCoherentSet_FnPtr layerBufferCoherentSet;
    layerBufferAllocate_FnPtr layerBufferAllocate;
    layerBufferIsAllocated_FnPtr layerBufferIsAllocated;
    layerBufferFree_FnPtr layerBufferFree;
    boolGet_FnPtr layerVisibleGet;
    boolSet_FnPtr layerVisibleSet;
    boolGet_FnPtr layerVsyncGet;
    boolSet_FnPtr layerVsyncSet;
    boolGet_FnPtr layerInvalidGet;
    boolSet_FnPtr layerInvalidSet;
    boolGet_FnPtr layerSwapSyncGet;
    boolSet_FnPtr layerSwapSyncSet;
    boolGet_FnPtr layerSwapGet;
    boolSet_FnPtr layerSwapSet;
    layerSwapPending_FnPtr layerSwapPending;
    layerSwapped_FnPtr layerSwapped;
    layerPositionGet_FnPtr layerPositionGet;
    layerPositionSet_FnPtr layerPositionSet;
    layerSizeGet_FnPtr layerSizeGet;
    layerSizeSet_FnPtr layerSizeSet;
    boolGet_FnPtr layerAlphaEnableGet;
    layerEffectSet_FnPtr layerAlphaEnableSet;
    layerAlphaAmountGet_FnPtr layerAlphaAmountGet;
    layerAlphaAmountSet_FnPtr layerAlphaAmountSet;
    layerMaskColorGet_FnPtr layerMaskEnableGet;
    layerMaskColorSet_FnPtr layerMaskEnableSet;
    boolGet_FnPtr layerMaskColorGet;
    layerEffectSet_FnPtr layerMaskColorSet;
    orientPoint_FnPtr orientPoint;
    mirrorPoint_FnPtr mirrorPoint;
    drawPipelineModeGet_FnPtr drawPipelineModeGet;
    drawPipelineModeSet_FnPtr drawPipelineModeSet;
    drawModeGet_FnPtr drawModeGet;
    drawModeSet_FnPtr drawModeSet;
    drawColorGet_FnPtr drawColorGet;
    drawColorSet_FnPtr drawColorSet;
    drawGradientColorGet_FnPtr drawGradientColorGet;
    drawGradientColorSet_FnPtr drawGradientColorSet;
    drawPaletteGet_FnPtr drawPaletteGet;
    drawPaletteSet_FnPtr drawPaletteSet;
    drawTargetGet_FnPtr drawTargetGet;
    drawTargetSet_FnPtr drawTargetSet;
    drawBlendModeGet_FnPtr drawBlendModeGet;
    drawBlendModeSet_FnPtr drawBlendModeSet;
    drawResizeModeGet_FnPtr drawResizeModeGet;
    drawResizeModeSet_FnPtr drawResizeModeSet;
    drawAlphaEnableGet_FnPtr drawAlphaEnableGet;
    drawAlphaEnableSet_FnPtr drawAlphaEnableSet;
    drawAlphaValueGet_FnPtr drawAlphaValueGet;
    drawAlphaValueSet_FnPtr drawAlphaValueSet;
    boolGet_FnPtr drawMaskEnableGet;
    boolSet_FnPtr drawMaskEnableSet;
    drawMaskValueGet_FnPtr drawMaskValueGet;
    drawMaskValueSet_FnPtr drawMaskValueSet;
    maskColor_FnPtr maskColor;
    boolGet_FnPtr drawAntialiasGet;
    boolSet_FnPtr drawAntialiasSet;
    drawThicknessGet_FnPtr drawThicknessGet;
    drawThicknessSet_FnPtr drawThicknessSet;
    boolGet_FnPtr drawClipEnableGet;
    boolSet_FnPtr drawClipEnableSet;
    drawClipRectGet_FnPtr drawClipRectGet;
    drawClipRectSet_FnPtr drawClipRectSet;
    blendGetPoint_FnPtr alphaGetPoint;
    blendColor_FnPtr alphaChannelBlend;
    blendColor_FnPtr globalAlphaBlend;
    GFX_DrawPipeline drawPipeline[GFX_PIPELINE_MODE_COUNT];
    interrupt_FnPtr interrupt;
} GFX_HAL;
```

Members

Members	Description
interrupt_FnPtr interrupt;	GFX_DRAW_PIPELINE_ENABLED

Description

Hardware Abstraction Function Table *

- This is the core hardware abtraction table that makes everything work. Drivers are
- expected to reroute the functionality of this table to hardware specific implementations
- to provide accelerated performance and features.

GFX_Handle Type

File

gfx_common.h

C

```
typedef void* GFX_Handle;
```

Description

This is type GFX_Handle.

GFX_Layer Structure

Layers describe basic display drawing areas and are meant to map directly to graphics controller hardware layers.

File

gfx_layer.h

C

```
typedef struct GFX_Layer_t {
  uint32_t id;
  struct {
    GFX_Rect display;
    GFX_Rect local;
  } rect;
  uint32_t pixel_size;
  GFX_Bool alphaEnable;
  uint32_t alphaAmount;
  GFX_Bool maskEnable;
  uint32_t maskColor;
  uint32_t buffer_count;
  uint32_t buffer_read_idx;
  uint32_t buffer_write_idx;
  GFX_FrameBuffer buffers[GFX_MAX_BUFFER_COUNT];
  GFX_Bool enabled;
  GFX_Bool visible;
  GFX_Bool swap;
  uint32_t swapCount;
  GFX_Bool locked;
  GFX_Bool invalid;
  GFX_Bool vsync;
  void* driver_data;
} GFX_Layer;
```

Members

Members	Description
GFX_Rect display;	represents area in display space
GFX_Rect local;	represents position in local space

Description

Structure: GFX_Layer_t

Graphics controllers will typically offer at least one drawing layer for drawing purposes. More advanced controllers may offer several. Layers are often configurable, offering independent positioning, sizing, color formats, and drawing features.

uint32_t id - the unique id of the layer

struct { GFX_Rect display - represents area in display space GFX_Rect local - represents position in local space } rect;

uint32_t pixel_size - size of a layer pixel in bytes

GFX_Bool alphaEnable - indicates if layer alpha blending is enabled uint32_t alphaAmount - indicates the amount of alpha blending

GFX_Bool maskEnable - indicates if layer masking/transparency is enabled uint32_t maskColor - the color to mask

uint32_t buffer_count - the number of buffers this layer owns uint32_t buffer_read_idx - the index of the current read buffer uint32_t buffer_write_idx - the index of the current write buffer GFX_FrameBuffer buffers[GFX_MAX_BUFFER_COUNT] - the layer buffer array

GFX_Bool enabled - indicates if the layer is enabled GFX_Bool visible - indicates if the layer is visible

GFX_Bool swap - indicates if the layer is waiting to advance its buffer chain

GFX_Bool locked - indicates if the layer's buffers are locked for manipulation. this is typically meant to prevent things like swapping before drawing operations have been completed

GFX_Bool vsync - indicates if this layer should swap during the display driver's blanking period. if this is true then it is the responsibility of the display driver to call GFX_LayerSwap on this layer during vblank. If this is false then the layer will swap immediately upon application request.

void* driver_data - this is a pointer that may be allocated by the display driver to store driver-specific per layer data. the driver is responsible for the management of this pointer during the application life cycle

Remarks

None.

GFX_Malloc_FnPtr Type

Simple wrapper around the standard malloc function pointer. Used for redirecting memory allocation to other memory management systems.

File

gfx_common.h

C

```
typedef void* (* GFX_Malloc_FnPtr)(size_t);
```

Description

memory abstraction

****

Function pointer

Function

typedef void* (*GFX_Malloc_FnPtr)(size_t);

GFX_Memcpy_FnPtr Type

Simple wrapper around the standard memcpy function pointer. Used for redirecting memcpy to other memory management systems.

File

gfx_common.h

C

```
typedef void* (* GFX_Memcpy_FnPtr)(void*, const void*, size_t);
```

Description

Function pointer

Function

typedef void* (*GFX_Memcpy_FnPtr)(void*, const void*, size_t);

GFX_MemoryIntf Structure

Custom memory manager interface.

File

gfx_common.h

C

```
typedef struct GFX_MemoryIntf_t {
  GFX_Malloc_FnPtr malloc;
  GFX_Malloc_FnPtr coherent_alloc;
  GFX_Calloc_FnPtr calloc;
  GFX_Realloc_FnPtr realloc;
  GFX_Free_FnPtr free;
  GFX_Free_FnPtr coherent_free;
  GFX_Memset_FnPtr memset;
  GFX_Memcpy_FnPtr memcpy;
} GFX_MemoryIntf;
```

Description

Structure: GFX_MemoryIntf_t

Applications utilizing the hardware abstraction layer may want to implement or utilize memory managers other than the standard library. This interface is the method for notifying the HAL of that manager.

The application must create a GFX_MemoryIntf struct, populate it with the fuction pointers that point to the custom memory manager, and pass the struct in to GFX_Open when the HAL context is created.

If no GFX_MemoryIntf is provided then the standard library memory management APIs will be used by default.

Remarks

None.

GFX_Memset_FnPtr Type

Simple wrapper around the standard memset function pointer. Used for redirecting memset to other memory management systems.

File

gfx_common.h

C

```
typedef void* (* GFX_Memset_FnPtr)(void*, int32_t, size_t);
```

Description

Function pointer

Function

typedef void* (*GFX_Memset_FnPtr)(void*, int32_t, size_t);

GFX_Orientation Enumeration

Orthogonal orientation settings.

File

gfx_common.h

C

```
typedef enum GFX_Orientation_t {
  GFX_ORIENTATION_0 = 0x0,
  GFX_ORIENTATION_90,
  GFX_ORIENTATION_180,
  GFX_ORIENTATION_270
} GFX_Orientation;
```

Description

Enumeration: GFX_Orientation_t

GFX_PixelBuffer Structure

A pixel buffer is a wrapper around a basic data pointer. A pixel buffer has a color mode, a pixel count, a rectangular dimension, a pixel count, and a lenght in bytes.

File

gfx_pixel_buffer.h

C

```
typedef struct GFX_PixelBuffer_t {
  GFX_ColorMode mode;
  GFX_Size size;
  int32_t pixel_count;
  uint32_t buffer_length;
  GFX_Buffer pixels;
} GFX_PixelBuffer;
```

Description

Structure: GFX_PixelBuffer_t

mode - the color mode of the pixel buffer size - the width and height dimension of the pixel buffer pixel_count - the total number of pixels in the buffer buffer_length - the total size of the buffer in bytes pixels - the pointer to the pixel data for the buffer

Remarks

None.

end_FnPtr Type

File

gfx_hal.h

C

```
typedef void (* end_FnPtr)(void);
```

Description

This is type end_FnPtr.

GFX_Processor Type

File

gfx_common.h

C

```
typedef uint32_t GFX_Processor;
```

Description

This is type GFX_Processor.

GFX_Realloc_FnPtr Type

Simple wrapper around the standard realloc function pointer. Used for redirecting memory allocation to other memory management systems.

File

gfx_common.h

C

```
typedef void* (* GFX_Realloc_FnPtr)(void*, size_t);
```

Description

Function pointer

Function

typedef void* (*GFX_Realloc_FnPtr)(void*, size_t);

GFX_Result Type

File

gfx_common.h

C

```
typedef int32_t GFX_Result;
```

Description

This is type GFX_Result.

GFX_Size Structure

A two dimensional indication of size. Values are signed but should never be negative.

File

gfx_common.h

C

```
typedef struct GFX_Size_t {
  int32_t width;
  int32_t height;
} GFX_Size;
```

Description

Structure: GFX_Size_t

GFX_SyncCallback_FnPtr Type

File

gfx_common.h

C

```
typedef void (* GFX_SyncCallback_FnPtr)(void);
```

Description

This is type GFX_SyncCallback_FnPtr.

layerActiveGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* layerActiveGet_FnPtr)(void);
```

Description

This is type layerActiveGet_FnPtr.

layerActiveSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerActiveSet_FnPtr)(uint32_t);
```

Description

This is type layerActiveSet_FnPtr.

layerAlphaAmountGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* layerAlphaAmountGet_FnPtr)(void);
```

Description

This is type layerAlphaAmountGet_FnPtr.

layerAlphaAmountSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerAlphaAmountSet_FnPtr)(uint32_t, GFX_Bool);
```

Description

This is type layerAlphaAmountSet_FnPtr.

layerBufferAddressGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Buffer (* layerBufferAddressGet_FnPtr)(uint32_t);
```

Description

This is type layerBufferAddressGet_FnPtr.

layerBufferAddressSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerBufferAddressSet_FnPtr)(uint32_t, GFX_Buffer);
```

Description

This is type layerBufferAddressSet_FnPtr.

layerBufferAllocate_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerBufferAllocate_FnPtr)(uint32_t);
```

Description

This is type layerBufferAllocate_FnPtr.

layerBufferCoherentGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Bool (* layerBufferCoherentGet_FnPtr)(uint32_t);
```

Description

This is type layerBufferCoherentGet_FnPtr.

layerBufferCoherentSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerBufferCoherentSet_FnPtr)(uint32_t, GFX_Bool);
```

Description

This is type layerBufferCoherentSet_FnPtr.

layerBufferCountGet_FnPtr Type

File

gfx_hal.h

C

```
typedef uint32_t (* layerBufferCountGet_FnPtr)(void);
```

Description

This is type layerBufferCountGet_FnPtr.

layerBufferCountSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerBufferCountSet_FnPtr)(uint32_t);
```

Description

This is type layerBufferCountSet_FnPtr.

layerBufferFree_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerBufferFree_FnPtr)(uint32_t);
```

Description

This is type layerBufferFree_FnPtr.

layerBufferIsAllocated_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Bool (* layerBufferIsAllocated_FnPtr)(uint32_t);
```

Description

This is type layerBufferIsAllocated_FnPtr.

layerMaskColorGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Color (* layerMaskColorGet_FnPtr)(void);
```

Description

This is type layerMaskColorGet_FnPtr.

layerMaskColorSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerMaskColorSet_FnPtr)(GFX_Color mask, GFX_Bool);
```

Description

This is type layerMaskColorSet_FnPtr.

layerPositionGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerPositionGet_FnPtr)(int32_t*, int32_t*);
```

Description

This is type layerPositionGet_FnPtr.

layerPositionSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerPositionSet_FnPtr)(int32_t, int32_t);
```

Description

This is type layerPositionSet_FnPtr.

layerSizeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerSizeGet_FnPtr)(int32_t* width, int32_t* height);
```

Description

This is type layerSizeGet_FnPtr.

layerSizeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerSizeSet_FnPtr)(int32_t width, int32_t height);
```

Description

This is type layerSizeSet_FnPtr.

layerSwapped_FnPtr Type

File

gfx_hal.h

C

```
typedef void (* layerSwapped_FnPtr)(GFX_Layer*);
```

Description

This is type layerSwapped_FnPtr.

orientationGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Orientation (* orientationGet_FnPtr)(void);
```

Description

This is type orientationGet_FnPtr.

orientationSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* orientationSet_FnPtr)(GFX_Orientation);
```

Description

This is type orientationSet_FnPtr.

pixelGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Color (* pixelGet_FnPtr)(const GFX_PixelBuffer*, const GFX_Point*);
```

Description

This is type pixelGet_FnPtr.

pixelSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* pixelSet_FnPtr)(const GFX_PixelBuffer*, const GFX_Point*, GFX_Color color);
```

Description

This is type pixelSet_FnPtr.

syncCallbackGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_SyncCallback_FnPtr (* syncCallbackGet_FnPtr)(void);
```

Description

This is type syncCallbackGet_FnPtr.

syncCallbackSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* syncCallbackSet_FnPtr)(GFX_SyncCallback_FnPtr);
```

Description

This is type syncCallbackSet_FnPtr.

syncCallbackSt_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* syncCallbackSt_FnPtr)(GFX_SyncCallback_FnPtr);
```

Description

This is type syncCallbackSt_FnPtr.

update_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* update_FnPtr)(void);
```

Description

This is type update_FnPtr.

GFX_ColorInfo Variable

File

gfx_color.h

C

```
LIB_EXPORT GFX_ColorModeInfo GFX_ColorInfo[GFX_COLOR_MODE_COUNT];
```

Description

This is variable GFX_ColorInfo.

GFX_Rect_Zero Variable

File

gfx_rect.h

C

```
const GFX_Rect GFX_Rect_Zero = {0, 0, 0, 0};
```

Description

This is variable GFX_Rect_Zero.

AGBA_8888_ALPHA_MASK Macro

File

gfx_color.h

C

```
#define AGBA_8888_ALPHA_MASK 0xFF000000
```

Description

This is macro AGBA_8888_ALPHA_MASK.

AGBA_8888_BLUE_MASK Macro

File

gfx_color.h

C

```
#define AGBA_8888_BLUE_MASK 0xFF
```

Description

This is macro AGBA_8888_BLUE_MASK.

AGBA_8888_GREEN_MASK Macro

File

gfx_color.h

C

```
#define AGBA_8888_GREEN_MASK 0xFF00
```

Description

This is macro AGBA_8888_GREEN_MASK.

AGBA_8888_RED_MASK Macro

File

gfx_color.h

C

```
#define AGBA_8888_RED_MASK 0xFF0000
```

Description

This is macro AGBA_8888_RED_MASK.

GFX_ANTIALIAS_MODE_COUNT Macro

File

gfx_draw.h

C

```
#define GFX_ANTIALIAS_MODE_COUNT (GFX_ANTIALIAS_ON+1)
```

Description

This is macro GFX_ANTIALIAS_MODE_COUNT.

GFX_COLOR_MAX_SIZE Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MAX_SIZE 4
```

Description

This is macro GFX_COLOR_MAX_SIZE.

GFX_COLOR_MODE_COUNT Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MODE_COUNT (GFX_COLOR_MODE_LAST + 1)
```

Description

This is macro GFX_COLOR_MODE_COUNT.

GFX_COLOR_MODE_IS_ALPHA Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MODE_IS_ALPHA(mode) ((mode == GFX_COLOR_MODE_RGBA_5551) || (mode ==
GFX_COLOR_MODE_RGBA_8888) || (mode == GFX_COLOR_MODE_ARGB_8888))
```

Description

This is macro GFX_COLOR_MODE_IS_ALPHA.

GFX_COLOR_MODE_IS_INDEX Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MODE_IS_INDEX(mode) ((mode >= GFX_COLOR_MODE_INDEX_1) && (mode <= GFX_COLOR_MODE_INDEX_8))
```

Description

This is macro GFX_COLOR_MODE_IS_INDEX.

GFX_COLOR_MODE_IS_PIXEL Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MODE_IS_PIXEL(mode) ((mode >= GFX_COLOR_MODE_GS_8) && (mode <= GFX_COLOR_MODE_YUV))
```

Description

This is macro GFX_COLOR_MODE_IS_PIXEL.

GFX_COLOR_MODE_LAST_COLOR Macro

File

gfx_color.h

C

```
#define GFX_COLOR_MODE_LAST_COLOR (GFX_COLOR_MODE_YUV)
```

Description

This is macro GFX_COLOR_MODE_LAST_COLOR.

GFX_DRAW_MODE_COUNT Macro

File

gfx_draw.h

C

```
#define GFX_DRAW_MODE_COUNT (GFX_DRAW_GRADIENT_TOP_BOTTOM + 1)
```

Description

This is macro GFX_DRAW_MODE_COUNT.

GFX_FAILURE Macro

File

gfx_common.h

C

```
#define GFX_FAILURE -1
```

Description

This is macro GFX_FAILURE.

GFX_FALSE Macro

File

gfx_common.h

C

```
#define GFX_FALSE 0
```

Description

This is macro GFX_FALSE.

GFX_MAX_BUFFER_COUNT Macro

File

gfx_common.h

C

```
#define GFX_MAX_BUFFER_COUNT 2
```

Description

This is macro GFX_MAX_BUFFER_COUNT.

GFX_NULL Macro

File

gfx_common.h

C

```
#define GFX_NULL 0
```

Description

This is macro GFX_NULL.

GFX_NUM_FLAGS Macro

File

gfx_common.h

C

```
#define GFX_NUM_FLAGS GFXF_LAST_FLAG
```

Description

This is macro GFX_NUM_FLAGS.

GFX_SUCCESS Macro

File

gfx_common.h

C

```
#define GFX_SUCCESS 0
```

Description

This is macro GFX_SUCCESS.

GFX_TRUE Macro

File

gfx_common.h

C

```
#define GFX_TRUE 1
```

Description

This is macro GFX_TRUE.

GFX_UNSUPPORTED Macro

File

gfx_common.h

C

```
#define GFX_UNSUPPORTED -2
```

Description

This is macro GFX_UNSUPPORTED.

RGB_2_BITS Macro

File

gfx_color.h

C

```
#define RGB_2_BITS 2
```

Description

This is macro RGB_2_BITS.

RGB_3_BITS Macro

File

gfx_color.h

C

```
#define RGB_3_BITS 7
```

Description

This is macro RGB_3_BITS.

RGB_332_BLUE_MASK Macro

File

gfx_color.h

C

```
#define RGB_332_BLUE_MASK 0x3
```

Description

This is macro RGB_332_BLUE_MASK.

RGB_332_GREEN_MASK Macro

File

gfx_color.h

C

```
#define RGB_332_GREEN_MASK 0x1C
```

Description

This is macro RGB_332_GREEN_MASK.

RGB_332_RED_MASK Macro

File

gfx_color.h

C

```
#define RGB_332_RED_MASK 0xE0
```

Description

This is macro RGB_332_RED_MASK.

RGB_5_BITS Macro

File

gfx_color.h

C

```
#define RGB_5_BITS 31
```

Description

This is macro RGB_5_BITS.

RGB_565_BLUE_MASK Macro

File

gfx_color.h

C

```
#define RGB_565_BLUE_MASK 0x1F
```

Description

This is macro RGB_565_BLUE_MASK.

RGB_565_GREEN_MASK Macro

File

gfx_color.h

C

```
#define RGB_565_GREEN_MASK 0x7E0
```

Description

This is macro RGB_565_GREEN_MASK.

RGB_565_RED_MASK Macro

File

gfx_color.h

C

```
#define RGB_565_RED_MASK 0xF800
```

Description

This is macro RGB_565_RED_MASK.

RGB_6_BITS Macro

File

gfx_color.h

C

```
#define RGB_6_BITS 63
```

Description

This is macro RGB_6_BITS.

RGB_8_BITS Macro

File

gfx_color.h

C

```
#define RGB_8_BITS 255
```

Description

This is macro RGB_8_BITS.

RGB_888_BLUE_MASK Macro

File

gfx_color.h

C

```
#define RGB_888_BLUE_MASK 0xFF
```

Description

This is macro RGB_888_BLUE_MASK.

RGB_888_GREEN_MASK Macro

File

gfx_color.h

C

```
#define RGB_888_GREEN_MASK 0xFF00
```

Description

This is macro RGB_888_GREEN_MASK.

RGB_888_RED_MASK Macro

File

gfx_color.h

C

```
#define RGB_888_RED_MASK 0xFF0000
```

Description

This is macro RGB_888_RED_MASK.

RGBA_5551_ALPHA_MASK Macro

File

gfx_color.h

C

```
#define RGBA_5551_ALPHA_MASK 0x1
```

Description

This is macro RGBA_5551_ALPHA_MASK.

RGBA_5551_BLUE_MASK Macro

File

gfx_color.h

C

```
#define RGBA_5551_BLUE_MASK 0x3E
```

Description

This is macro RGBA_5551_BLUE_MASK.

RGBA_5551_GREEN_MASK Macro

File

gfx_color.h

C

```
#define RGBA_5551_GREEN_MASK 0x7C0
```

Description

This is macro RGBA_5551_GREEN_MASK.

RGBA_5551_RED_MASK Macro

File

gfx_color.h

C

```
#define RGBA_5551_RED_MASK 0xF800
```

Description

This is macro RGBA_5551_RED_MASK.

RGBA_8888_ALPHA_MASK Macro

File

gfx_color.h

C

```
#define RGBA_8888_ALPHA_MASK 0xFF
```

Description

This is macro RGBA_8888_ALPHA_MASK.

RGBA_8888_BLUE_MASK Macro

File

gfx_color.h

C

```
#define RGBA_8888_BLUE_MASK 0xFF00
```

Description

This is macro RGBA_8888_BLUE_MASK.

RGBA_8888_GREEN_MASK Macro

File

gfx_color.h

C

```
#define RGBA_8888_GREEN_MASK 0xFF0000
```

Description

This is macro RGBA_8888_GREEN_MASK.

RGBA_8888_RED_MASK Macro

File

gfx_color.h

C

```
#define RGBA_8888_RED_MASK 0xFF000000
```

Description

This is macro RGBA_8888_RED_MASK.

initialize_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* initialize_FnPtr)(GFX_Context*);
```

Description

This is type initialize_FnPtr.

interrupt_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* interrupt_FnPtr)(uint32_t);
```

Description

GFX_DRAW_PIPELINE_ENABLED

blendColor_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Color (* blendColor_FnPtr)(GFX_Color source, GFX_Color dest, GFX_ColorMode mode);
```

Description

This is type blendColor_FnPtr.

blendGetPoint_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Color (* blendGetPoint_FnPtr)(const GFX_PixelBuffer* buffer, const GFX_Point* pnt, const
GFX_DrawState* state);
```

Description

This is type blendGetPoint_FnPtr.

drawPipelineModeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_PipelineMode (* drawPipelineModeGet_FnPtr)(void);
```

Description

This is type drawPipelineModeGet_FnPtr.

drawPipelineModeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawPipelineModeSet_FnPtr)(GFX_PipelineMode);
```

Description

This is type drawPipelineModeSet_FnPtr.

drawResizeModeGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_ResizeMode (* drawResizeModeGet_FnPtr)(void);
```

Description

This is type drawResizeModeGet_FnPtr.

drawResizeModeSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawResizeModeSet_FnPtr)(GFX_ResizeMode);
```

Description

This is type drawResizeModeSet_FnPtr.

drawStretchBlit_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawStretchBlit_FnPtr)(const GFX_PixelBuffer*, const GFX_Rect*, const GFX_Rect*,
const GFX_DrawState*);
```

Description

This is type drawStretchBlit_FnPtr.

drawTargetGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawTargetGet_FnPtr)(GFX_PixelBuffer**);
```

Description

This is type drawTargetGet_FnPtr.

drawTargetSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* drawTargetSet_FnPtr)(const GFX_PixelBuffer*);
```

Description

This is type drawTargetSet_FnPtr.

GFX_DrawPipeline Structure

File

gfx_hal.h

C

```
typedef struct GFX_DrawPipeline_t {
  drawPixel_FnPtr drawPixel[GFX_ANTIALIAS_MODE_COUNT];
  drawLine_FnPtr drawLine[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawLine_FnPtr drawHorzLine[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawLine_FnPtr drawVertLine[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawRect_FnPtr drawRect[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawCircle_FnPtr drawCircle[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawArc_FnPtr drawArc[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawEllipse_FnPtr drawEllipse[GFX_DRAW_MODE_COUNT][GFX_ANTIALIAS_MODE_COUNT];
  drawBlit_FnPtr drawBlit;
  drawStretchBlit_FnPtr drawStretchBlit[GFX_RESIZE_MODE_COUNT];
  drawBlit_FnPtr drawDirectBlit;
  pixelGet_FnPtr pixelGet;
  pixelGetArray_FnPtr pixelArrayGet;
  pixelSet_FnPtr pixelSet;
} GFX_DrawPipeline;
```

Members

Members	Description
drawPixel_FnPtr drawPixel[GFX_ANTIALIAS_MODE_COUNT];	draw functions

Section

Data Types and Constants

GFX_PipelineMode Enumeration

Hardware draw path settings.

File

gfx_common.h

C

```
typedef enum GFX_PipelineMode_t {
  GFX_PIPELINE_SOFTWARE = 0,
  GFX_PIPELINE_GCU = 1,
  GFX_PIPELINE_GPU = 2,
  GFX_PIPELINE_GCUGPU = 3
} GFX_PipelineMode;
```

Section

Data Types and Constants

****

****

****

Enumeration:

GFX_HardwareMode_t

GFX_ResizeMode Enumeration

File

gfx_draw.h

C

```
typedef enum GFX_ResizeMode_t {
  GFX_RESIZE_NEARESTNEIGHBOR = 0x0,
  GFX_RESIZE_BILINEAR
} GFX_ResizeMode;
```

Description

This is type GFX_ResizeMode.

GFXU_ImageFlags Enumeration

A list of flags describing an image asset

File

gfxu_image.h

C

```
typedef enum GFXU_ImageFlags_t {
  GFXU_IMAGE_USE_MASK = 0x1,
  GFXU_IMAGE_SUPPORTS_CLIPPING = 0x2,
  GFXU_IMAGE_DIRECT_BLIT = 0x4
} GFXU_ImageFlags;
```

Description

Enumeration: GFXU_ImageFlags_t

layerSwapPending_FnPtr Type

File

gfx_hal.h

C

```
typedef void (* layerSwapPending_FnPtr)(GFX_Layer*);
```

Description

This is type layerSwapPending_FnPtr.

maskColor_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Bool (* maskColor_FnPtr)(const GFX_DrawState* state, GFX_Color color);
```

Description

This is type maskColor_FnPtr.

mirrorPoint_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Point (* mirrorPoint_FnPtr)(const GFX_Point* pnt, const GFX_Rect* rect, GFX_Orientation ori);
```

Description

This is type mirrorPoint_FnPtr.

orientPoint_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Point (* orientPoint_FnPtr)(const GFX_Point* pnt, const GFX_Rect* rect, GFX_Orientation ori);
```

Description

This is type orientPoint_FnPtr.

pixelGetArray_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* pixelGetArray_FnPtr)(GFX_BufferSelection source, const GFX_Rect*, GFX_PixelBuffer*);
```

Description

This is type pixelGetArray_FnPtr.

GFX_PIPELINE_MODE_COUNT Macro

File

gfx_common.h

C

```
#define GFX_PIPELINE_MODE_COUNT (GFX_PIPELINE_GCUGPU + 1)
```

Description

This is macro GFX_PIPELINE_MODE_COUNT.

GFX_RESIZE_MODE_COUNT Macro

File

gfx_draw.h

C

```
#define GFX_RESIZE_MODE_COUNT (GFX_RESIZE_BILINEAR+1)
```

Description

This is macro GFX_RESIZE_MODE_COUNT.

GFX_ASSERT Macro

File

gfx_common.h

C

```
#define GFX_ASSERT(x) { }
```

Description

This is macro GFX_ASSERT.

GFX_GLOBAL_PALETTE_SIZE Macro

File

gfx_common.h

C

```
#define GFX_GLOBAL_PALETTE_SIZE 256
```

Description

This is macro GFX_GLOBAL_PALETTE_SIZE.

GFX_GlobalPalette Type

File

gfx_common.h

C

```
typedef void* GFX_GlobalPalette;
```

Description

This is type GFX_GlobalPalette.

globalPaletteGet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_GlobalPalette (* globalPaletteGet_FnPtr)(void);
```

Description

This is type globalPaletteGet_FnPtr.

globalPaletteSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* globalPaletteSet_FnPtr)(GFX_GlobalPalette);
```

Description

This is type globalPaletteSet_FnPtr.

layerEffectSet_FnPtr Type

File

gfx_hal.h

C

```
typedef GFX_Result (* layerEffectSet_FnPtr)(GFX_Bool, GFX_Bool);
```

Description

This is type layerEffectSet_FnPtr.

GFX_DEPRECATED Macro

File

gfx_common.h

C

```
#define GFX_DEPRECATED __attribute__ ((deprecated))
```

Description

This is macro GFX_DEPRECATED.

Files

Files

Name	Description
gfx_common.h	Common definitions for MPLAB Harmony Graphics Hardware Abstraction Layer
gfx_color.h	Contains functions for color information and manipulation operations
gfx_context.h	Defines MPLAB Harmony Graphics Hardware Abstraction Layer Context
gfx_default_impl.h	This is file gfx_default_impl.h.
gfx_display.h	Defines MPLAB Harmony Graphics Hardware Abstraction Layer display information struct
gfx_draw.h	Main header file for MPLAB Harmony Graphics Hardware Abstraction primitive draw functions
gfx_draw_blit.h	This is file gfx_draw_blit.h.
gfx_draw_circle.h	This is file gfx_draw_circle.h.
gfx_draw_line.h	This is file gfx_draw_line.h.
gfx_draw_pixel.h	This is file gfx_draw_pixel.h.
gfx_draw_rect.h	This is file gfx_draw_rect.h.
gfx_driver_interface.h	Defines MPLAB Harmony Graphics Hardware Abstraction Layer driver interface struct
gfx_hal.h	This is file gfx_hal.h.
gfx_interface.h	This is file gfx_interface.h.
gfx_layer.h	Defines the graphics layer construct
gfx_math.h	Contains some general purpose math functions
gfx_pixel_buffer.h	Defines a general purpose pixel buffer construct.
gfx_rect.h	Defines general purposes rectangle functions.
gfx_util.h	Utility functions for the Hardware Abstraction Layer

Description

gfx_common.h

Common definitions for MPLAB Harmony Graphics Hardware Abstraction Layer

Enumerations

	Name	Description
	GFX_BlendMode_t	Blend mode masks

	GFX_BufferSelection_t	Buffer selector used when querying layers for certain buffer states.
	GFX_BufferState_t	Frame buffer memory states
	GFX_Flag_t	Hardware abstraction state interface flags. See gfx.h for a comprehensive description.
	GFX_Orientation_t	Orthogonal orientation settings.
	GFX_PipelineMode_t	Hardware draw path settings.
	GFX_BlendMode	Blend mode masks
	GFX_BufferSelection	Buffer selector used when querying layers for certain buffer states.
	GFX_BufferState	Frame buffer memory states
	GFX_Flag	Hardware abstraction state interface flags. See gfx.h for a comprehensive description.
	GFX_Orientation	Orthogonal orientation settings.
	GFX_PipelineMode	Hardware draw path settings.

Macros

	Name	Description
	GFX_ASSERT	This is macro GFX_ASSERT.
	GFX_DEPRECATED	This is macro GFX_DEPRECATED.
	GFX_FAILURE	This is macro GFX_FAILURE.
	GFX_FALSE	This is macro GFX_FALSE.
	GFX_GLOBAL_PALETTE_SIZE	This is macro GFX_GLOBAL_PALETTE_SIZE.
	GFX_MAX_BUFFER_COUNT	This is macro GFX_MAX_BUFFER_COUNT.
	GFX_NULL	This is macro GFX_NULL.
	GFX_NUM_FLAGS	This is macro GFX_NUM_FLAGS.
	GFX_PIPELINE_MODE_COUNT	This is macro GFX_PIPELINE_MODE_COUNT.
	GFX_SUCCESS	This is macro GFX_SUCCESS.
	GFX_TRUE	This is macro GFX_TRUE.
	GFX_UNSUPPORTED	This is macro GFX_UNSUPPORTED.

Structures

	Name	Description
	GFX_MemoryIntf_t	Custom memory manager interface.
	GFX_Point_t	A two dimensional Cartesian point.
	GFX_Rect_t	A rectangle definition.
	GFX_Size_t	A two dimensional indication of size. Values are signed but should never be negative.
	GFX_MemoryIntf	Custom memory manager interface.
	GFX_Size	A two dimensional indication of size. Values are signed but should never be negative.

Types

	Name	Description
	GFX_Bool	This is type GFX_Bool.
	GFX_Buffer	This is type GFX_Buffer.
	GFX_Calloc_FnPtr	Simple wrapper around the standard calloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Color	This is type GFX_Color.
	GFX_ColorMode	List of available color modes.
	GFX_Display	This is type GFX_Display.
	GFX_Driver	This is type GFX_Driver.
	GFX_Free_FnPtr	Simple wrapper around the standard free function pointer. Used for redirecting memory free to other memory management systems.
	GFX_GlobalPalette	This is type GFX_GlobalPalette.
	GFX_Handle	This is type GFX_Handle.
	GFX_Malloc_FnPtr	Simple wrapper around the standard malloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Memcpy_FnPtr	Simple wrapper around the standard memcpy function pointer. Used for redirecting memcpy to other memory management systems.
	GFX_Memset_FnPtr	Simple wrapper around the standard memset function pointer. Used for redirecting memset to other memory management systems.
	GFX_Processor	This is type GFX_Processor.

	GFX_Realloc_FnPtr	Simple wrapper around the standard realloc function pointer. Used for redirecting memory allocation to other memory management systems.
	GFX_Result	This is type GFX_Result.
	GFX_SyncCallback_FnPtr	This is type GFX_SyncCallback_FnPtr.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Type definitions for common functions.

File Name

gfx_common.h

Company

Microchip Technology Inc.

gfx_color.h

Contains functions for color information and manipulation operations

Enumerations

	Name	Description
	GFX_BitsPerPixel_t	List of available bits-per-pixel sizes.
	GFX_ColorMask_t	Maskable list of color valies.
	GFX_ColorMode_t	List of available color modes.
	GFX_ColorName_t	Color name reference table
	GFX_BitsPerPixel	List of available bits-per-pixel sizes.
	GFX_ColorMask	Maskable list of color valies.
	GFX_ColorName	Color name reference table

Functions

	Name	Description
	GFX_ColorBilerp	Calculates bilinear interpolation between four colors
	GFX_ColorBlend_RGBA_8888	Blends two RGBA8888 colors together using their alpha channel values.
	GFX_ColorChannelAlpha	Used for getting the alpha color channel of a given color value.
	GFX_ColorChannelBlue	Used for getting the blue color channel of a given color value.
	GFX_ColorChannelGreen	Used for getting the green color channel of a given color value.
	GFX_ColorChannelRed	Used for getting the red color channel of a given color value.
	GFX_ColorConvert	Converts a color value from one mode to another
	GFX_ColorLerp	Linear interpolation between two colors
	GFX_ColorModeInfoGet	
	GFX_ColorValue	Used for getting a color value by name.

Macros

	Name	Description
	AGBA_8888_ALPHA_MASK	This is macro AGBA_8888_ALPHA_MASK.
	AGBA_8888_BLUE_MASK	This is macro AGBA_8888_BLUE_MASK.
	AGBA_8888_GREEN_MASK	This is macro AGBA_8888_GREEN_MASK.
	AGBA_8888_RED_MASK	This is macro AGBA_8888_RED_MASK.
	GFX_COLOR_MAX_SIZE	This is macro GFX_COLOR_MAX_SIZE.
	GFX_COLOR_MODE_COUNT	This is macro GFX_COLOR_MODE_COUNT.
	GFX_COLOR_MODE_IS_ALPHA	This is macro GFX_COLOR_MODE_IS_ALPHA.
	GFX_COLOR_MODE_IS_INDEX	This is macro GFX_COLOR_MODE_IS_INDEX.
	GFX_COLOR_MODE_IS_PIXEL	This is macro GFX_COLOR_MODE_IS_PIXEL.
	GFX_COLOR_MODE_LAST_COLOR	This is macro GFX_COLOR_MODE_LAST_COLOR.
	RGB_2_BITS	This is macro RGB_2_BITS.
	RGB_3_BITS	This is macro RGB_3_BITS.
	RGB_332_BLUE_MASK	This is macro RGB_332_BLUE_MASK.

	RGB_332_GREEN_MASK	This is macro RGB_332_GREEN_MASK.
	RGB_332_RED_MASK	This is macro RGB_332_RED_MASK.
	RGB_5_BITS	This is macro RGB_5_BITS.
	RGB_565_BLUE_MASK	This is macro RGB_565_BLUE_MASK.
	RGB_565_GREEN_MASK	This is macro RGB_565_GREEN_MASK.
	RGB_565_RED_MASK	This is macro RGB_565_RED_MASK.
	RGB_6_BITS	This is macro RGB_6_BITS.
	RGB_8_BITS	This is macro RGB_8_BITS.
	RGB_888_BLUE_MASK	This is macro RGB_888_BLUE_MASK.
	RGB_888_GREEN_MASK	This is macro RGB_888_GREEN_MASK.
	RGB_888_RED_MASK	This is macro RGB_888_RED_MASK.
	RGBA_5551_ALPHA_MASK	This is macro RGBA_5551_ALPHA_MASK.
	RGBA_5551_BLUE_MASK	This is macro RGBA_5551_BLUE_MASK.
	RGBA_5551_GREEN_MASK	This is macro RGBA_5551_GREEN_MASK.
	RGBA_5551_RED_MASK	This is macro RGBA_5551_RED_MASK.
	RGBA_8888_ALPHA_MASK	This is macro RGBA_8888_ALPHA_MASK.
	RGBA_8888_BLUE_MASK	This is macro RGBA_8888_BLUE_MASK.
	RGBA_8888_GREEN_MASK	This is macro RGBA_8888_GREEN_MASK.
	RGBA_8888_RED_MASK	This is macro RGBA_8888_RED_MASK.

Structures

	Name	Description
	GFX_ColorModeInfo_t	Struct that provides information about a color mode.
	GFX_ColorModeInfo	Struct that provides information about a color mode.

Variables

	Name	Description
	GFX_ColorInfo	This is variable GFX_ColorInfo.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Color conversion and color channel management

File Name

gfx_color.h

Company

Microchip Technology Inc.

gfx_context.h

Defines MPLAB Harmony Graphics Hardware Abstraction Layer Context

Functions

	Name	Description
	GFX_ActiveContext	Gets the current set active HAL context.
	GFX_ContextActiveSet	Sets the active context

Structures

	Name	Description
	GFX_Context_t	An instance of the hardware abstraction layer.
	GFX_Context	An instance of the hardware abstraction layer.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

HAL context management functions.

File Name

gfx_context.h

Company

Microchip Technology Inc.

gfx_default_impl.h

This is file gfx_default_impl.h.

gfx_display.h

Defines MPLAB Harmony Graphics Hardware Abstraction Layer display information struct

Structures

	Name	Description
	GFX_DisplayInfo_t	Describes a graphical display device.
	GFX_DisplayInfo	Describes a graphical display device.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer
Display information.

File Name

gfx_display.h

Company

Microchip Technology Inc.

gfx_draw.h

Main header file for MPLAB Harmony Graphics Hardware Abstraction primitive draw functions

Enumerations

	Name	Description
	GFX_AntialiasMode_t	Enables anti-aliased drawing hint
	GFX_DrawMode_t	Gradient draw modes.
	GFX_ResizeMode_t	This is type GFX_ResizeMode.
	GFX_AntialiasMode	Enables anti-aliased drawing hint
	GFX_DrawMode	Gradient draw modes.
	GFX_ResizeMode	This is type GFX_ResizeMode.

Functions

	Name	Description
	GFX_DrawBlit	Blits a buffer of pixels into the frame buffer.
	GFX_DrawCircle	Draws a circle from using the specified dimensions and the current draw state.
	GFX_DrawDirectBlit	Blits a buffer of pixels into the frame buffer without performing per-pixel operations on the data.
	GFX_DrawLine	Draws a line from (x1,y1) to (x2,y2) using the current draw state.
	GFX_DrawPixel	Sets the pixel at X and Y using the current draw state.
	GFX_DrawRect	Draws a rectangle using the specified dimensions and the current draw state.
	GFX_DrawStretchBlit	Blits a buffer of pixels into the frame buffer.

Macros

	Name	Description
	GFX_ANTIALIAS_MODE_COUNT	This is macro GFX_ANTIALIAS_MODE_COUNT.
	GFX_DRAW_MODE_COUNT	This is macro GFX_DRAW_MODE_COUNT.
	GFX_RESIZE_MODE_COUNT	This is macro GFX_RESIZE_MODE_COUNT.

Structures

	Name	Description
	GFX_DrawState_t	A list of drawing hints for shape drawing algorithms
	GFX_DrawState	A list of drawing hints for shape drawing algorithms

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Shape drawing functions.

File Name

gfx_draw.h

Company

Microchip Technology Inc.

gfx_draw_blit.h

This is file gfx_draw_blit.h.

gfx_draw_circle.h

This is file gfx_draw_circle.h.

gfx_draw_line.h

This is file gfx_draw_line.h.

gfx_draw_pixel.h

This is file gfx_draw_pixel.h.

gfx_draw_rect.h

This is file gfx_draw_rect.h.

gfx_driver_interface.h

Defines MPLAB Harmony Graphics Hardware Abstraction Layer driver interface struct

Structures

	Name	Description
	GFX_DriverInfo_t	A driver description structure.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Display driver information, internal use.

File Name

gfx_draw_interface.h

Company

Microchip Technology Inc.

gfx_hal.h

Structures

	Name	Description
	GFX_DrawPipeline_t	
	GFX_HAL_t	Hardware Abstraction Function Table * • This is the core hardware abtraction table that makes everything work. Drivers are • expected to reroute the functionality of this table to hardware specific implementations • to provide accelerated performance and features.
	GFX_DrawPipeline	
	GFX_HAL	Hardware Abstraction Function Table * • This is the core hardware abtraction table that makes everything work. Drivers are • expected to reroute the functionality of this table to hardware specific implementations • to provide accelerated performance and features.

Types

	Name	Description
	begin_FnPtr	This is type begin_FnPtr.
	blendColor_FnPtr	This is type blendColor_FnPtr.
	blendGetPoint_FnPtr	This is type blendGetPoint_FnPtr.
	boolGet_FnPtr	This is type boolGet_FnPtr.
	boolSet_FnPtr	This is type boolSet_FnPtr.
	brightnessGet_FnPtr	This is type brightnessGet_FnPtr.
	brightnessRangeGet_FnPtr	This is type brightnessRangeGet_FnPtr.
	brightnessSet_FnPtr	This is type brightnessSet_FnPtr.
	colorModeGet_FnPtr	This is type colorModeGet_FnPtr.
	colorModeSet_FnPtr	This is type colorModeSet_FnPtr.
	destroy_FnPtr	This is type destroy_FnPtr.
	drawAlphaValueGet_FnPtr	This is type drawAlphaValueGet_FnPtr.
	drawAlphaValueSet_FnPtr	This is type drawAlphaValueSet_FnPtr.
	drawBlendModeGet_FnPtr	This is type drawBlendModeGet_FnPtr.
	drawBlendModeSet_FnPtr	This is type drawBlendModeSet_FnPtr.
	drawBlit_FnPtr	This is type drawBlit_FnPtr.
	drawCircle_FnPtr	This is type drawCircle_FnPtr.
	drawClipRectGet_FnPtr	This is type drawClipRectGet_FnPtr.
	drawClipRectSet_FnPtr	This is type drawClipRectSet_FnPtr.
	drawColorGet_FnPtr	This is type drawColorGet_FnPtr.
	drawColorSet_FnPtr	This is type drawColorSet_FnPtr.
	drawGradientColorGet_FnPtr	This is type drawGradientColorGet_FnPtr.
	drawGradientColorSet_FnPtr	This is type drawGradientColorSet_FnPtr.
	drawLine_FnPtr	This is type drawLine_FnPtr.
	drawLock_FnPtr	This is type drawLock_FnPtr.
	drawMaskValueGet_FnPtr	This is type drawMaskValueGet_FnPtr.
	drawMaskValueSet_FnPtr	This is type drawMaskValueSet_FnPtr.
	drawModeGet_FnPtr	This is type drawModeGet_FnPtr.
	drawModeSet_FnPtr	This is type drawModeSet_FnPtr.
	drawPaletteGet_FnPtr	This is type drawPaletteGet_FnPtr.
	drawPaletteSet_FnPtr	This is type drawPaletteSet_FnPtr.
	drawPipelineModeGet_FnPtr	This is type drawPipelineModeGet_FnPtr.
	drawPipelineModeSet_FnPtr	This is type drawPipelineModeSet_FnPtr.
	drawPixel_FnPtr	This is type drawPixel_FnPtr.
	drawRect_FnPtr	This is type drawRect_FnPtr.
	drawResizeModeGet_FnPtr	This is type drawResizeModeGet_FnPtr.
	drawResizeModeSet_FnPtr	This is type drawResizeModeSet_FnPtr.

	drawStretchBlit_FnPtr	This is type drawStretchBlit_FnPtr.
	drawTargetGet_FnPtr	This is type drawTargetGet_FnPtr.
	drawTargetSet_FnPtr	This is type drawTargetSet_FnPtr.
	drawThicknessGet_FnPtr	This is type drawThicknessGet_FnPtr.
	drawThicknessSet_FnPtr	This is type drawThicknessSet_FnPtr.
	drawUnlock_FnPtr	This is type drawUnlock_FnPtr.
	end_FnPtr	This is type end_FnPtr.
	GFX_DriverInfo	A driver description structure.
	globalPaletteGet_FnPtr	This is type globalPaletteGet_FnPtr.
	globalPaletteSet_FnPtr	This is type globalPaletteSet_FnPtr.
	initialize_FnPtr	This is type initialize_FnPtr.
	interrupt_FnPtr	GFX_DRAW_PIPELINE_ENABLED
	layerActiveGet_FnPtr	This is type layerActiveGet_FnPtr.
	layerActiveSet_FnPtr	This is type layerActiveSet_FnPtr.
	layerAlphaAmountGet_FnPtr	This is type layerAlphaAmountGet_FnPtr.
	layerAlphaAmountSet_FnPtr	This is type layerAlphaAmountSet_FnPtr.
	layerBufferAddressGet_FnPtr	This is type layerBufferAddressGet_FnPtr.
	layerBufferAddressSet_FnPtr	This is type layerBufferAddressSet_FnPtr.
	layerBufferAllocate_FnPtr	This is type layerBufferAllocate_FnPtr.
	layerBufferCoherentGet_FnPtr	This is type layerBufferCoherentGet_FnPtr.
	layerBufferCoherentSet_FnPtr	This is type layerBufferCoherentSet_FnPtr.
	layerBufferCountGet_FnPtr	This is type layerBufferCountGet_FnPtr.
	layerBufferCountSet_FnPtr	This is type layerBufferCountSet_FnPtr.
	layerBufferFree_FnPtr	This is type layerBufferFree_FnPtr.
	layerBufferIsAllocated_FnPtr	This is type layerBufferIsAllocated_FnPtr.
	layerEffectSet_FnPtr	This is type layerEffectSet_FnPtr.
	layerMaskColorGet_FnPtr	This is type layerMaskColorGet_FnPtr.
	layerMaskColorSet_FnPtr	This is type layerMaskColorSet_FnPtr.
	layerPositionGet_FnPtr	This is type layerPositionGet_FnPtr.
	layerPositionSet_FnPtr	This is type layerPositionSet_FnPtr.
	layerSizeGet_FnPtr	This is type layerSizeGet_FnPtr.
	layerSizeSet_FnPtr	This is type layerSizeSet_FnPtr.
	layerSwapped_FnPtr	This is type layerSwapped_FnPtr.
	layerSwapPending_FnPtr	This is type layerSwapPending_FnPtr.
	maskColor_FnPtr	This is type maskColor_FnPtr.
	mirrorPoint_FnPtr	This is type mirrorPoint_FnPtr.
	orientationGet_FnPtr	This is type orientationGet_FnPtr.
	orientationSet_FnPtr	This is type orientationSet_FnPtr.
	orientPoint_FnPtr	This is type orientPoint_FnPtr.
	pixelGet_FnPtr	This is type pixelGet_FnPtr.
	pixelGetArray_FnPtr	This is type pixelGetArray_FnPtr.
	pixelSet_FnPtr	This is type pixelSet_FnPtr.
	syncCallbackGet_FnPtr	This is type syncCallbackGet_FnPtr.
	syncCallbackSet_FnPtr	This is type syncCallbackSet_FnPtr.
	syncCallbackSt_FnPtr	This is type syncCallbackSt_FnPtr.
	update_FnPtr	This is type update_FnPtr.

Description

This is file gfx_hal.h.

gfx_interface.h

This is file gfx_interface.h.

gfx_layer.h

Defines the graphics layer construct

Functions

	Name	Description
	GFX_LayerFromOrientedSpace	Transforms a layer oriented space to screen space.
	GFX_LayerPointFromOrientedSpace	Transforms a point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerPointToOrientedSpace	Transforms a point from screen space to oriented space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerReadBuffer	Gets the pointer to the layer's current read pixel buffer.
	GFX_LayerRectFromOrientedSpace	Transforms a layer point from oriented space to screen space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerRectToOrientedSpace	Transforms a rectangle from screen space to oriented space given a layer, a display orientation, and a mirroring setting.
	GFX_LayerRotate	Swaps the width and height dimensions of a layer. Can be used for run-time display orientation
	GFX_LayerSwap	Performs a swap operation on the given layer. This advances the pointers of layer's buffer chain. The current write buffer becomes the new read buffer and a new buffer is chosen as the new write buffer. Has no effect in single buffer environments.
	GFX_LayerToOrientedSpace	Transforms a layer from screen space to oriented space.
	GFX_LayerWriteBuffer	Gets the pointer to the layer's current write pixel buffer.

Structures

	Name	Description
	GFX_FrameBuffer_t	A frame buffer is a wrapper around a pixel buffer construct that is used by display drivers to manage frame buffers.
	GFX_Layer_t	Layers describe basic display drawing areas and are meant to map directly to graphics controller hardware layers.
	GFX_FrameBuffer	A frame buffer is a wrapper around a pixel buffer construct that is used by display drivers to manage frame buffers.
	GFX_Layer	Layers describe basic display drawing areas and are meant to map directly to graphics controller hardware layers.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Layer and buffer management.

File Name

gfx_layer.h

Company

Microchip Technology Inc.

gfx_math.h

Contains some general purpose math functions

Functions

	Name	Description
	GFX_AbsoluteValue	Calculates the absolute value of a signed integer.
	GFX_Clampf	Clamps a float between a min and max
	GFX_Clampi	Clamps an integer between a min and max
	GFX_DivideRounding	This is function GFX_DivideRounding.
	GFX_Lerp	Performs a linear interpolation of an integer based on a percentage between two signed points.
	GFX_Maxf	Returns the larger of two floats.
	GFX_Maxi	Returns the larger of two integers.

	GFX_Minf	Returns the smaller of two floats.
	GFX_Mini	Returns the smaller of two integers.
	GFX_Percent	Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed. The result is the decimal percentage multiplied by 100.
	GFX_PercentOf	Calculates the percentage of a number. Returns a whole number with no decimal component.
	GFX_PercentOfDec	This is function GFX_PercentOfDec.
	GFX_PercentWholeRounded	Calculates the percentage of one number when applied to another. Integer based. Accuracy for higher numbers is not guaranteed. The difference between this and GFX_Percent is that the decimal portion of the whole number is rounded off.
	GFX_ScaleInteger	Scales an integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.
	GFX_ScaleIntegerSigned	Scales a signed integer from one number range of 0 -> n0 to another range 0 -> n1 based on percentages.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Math support functions.

File Name

gfx_layer.h

Company

Microchip Technology Inc.

gfx_pixel_buffer.h

Defines a general purpose pixel buffer construct.

Functions

	Name	Description
	GFX_PixelBufferAreaFill	Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer.
	GFX_PixelBufferAreaFill_Unsafe	Fills an area of a pixel buffer with a solid color. Caller is responsible for ensuring that the color is the same color format as the destination buffer. Like GFX_PixelBufferAreaFill but performs no bounds checking.
	GFX_PixelBufferAreaGet	Extracts a rectanglar section of pixels from a pixel buffer.
	GFX_PixelBufferAreaGet_Unsafe	Extracts a rectanglar section of pixels from a pixel buffer. Like GFX_PixelBufferAreaGet but performs no clipping between the rectangles of the extract area and the source buffer.
	GFX_PixelBufferAreaSet	Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation.
	GFX_PixelBufferAreaSet_Unsafe	Copies an area of pixels from a source buffer to a destination buffer. If the source buffer format does not match the destination format the data will be converted to match during the copy operation. Like GFX_PixelBufferAreaSet but performs no bounds checking.
	GFX_PixelBufferClipRect	Clips a rectangle against a pixel buffer. The result is guaranteed to fit inside the buffer's area.
	GFX_PixelBufferConvert	Duplicates a pixel buffer and converts the copy to another color mode.
	GFX_PixelBufferCopy	Creates a copy of the input buffer. If the input buffer's pixel data is not null then the data will be duplicated for the result buffer.
	GFX_PixelBufferCreate	Initializes a pixel buffer struct. Does not actually allocate any memory.
	GFX_PixelBufferDestroy	Destroys a pixel buffer construct. If the buffer's pixels pointer is not zero this function will attempt to free it using the provided GFX_MemoryIntf memory interface.
	GFX_PixelBufferGet	Gets the value of the pixel that resides at the provided point in the given buffer.
	GFX_PixelBufferGet_Unsafe	Gets the value of the pixel that resides at the provided point in the given buffer. Like GFX_PixelBufferGet but performs no bounds checking.
	GFX_PixelBufferGetIndex	Interprets the pixel buffer as a table of indices and looks up a specific index at position 'idx'. Indices may be 1bpp, 4bpp, or 8bpp in size and are indicated by the color mode of the pixel buffer.
	GFX_PixelBufferOffsetGet	Gets the offset address of the pixel that resides at the provided point in the given buffer.
	GFX_PixelBufferOffsetGet_Unsafe	Gets the offset address of the pixel that resides at the provided point in the given buffer. Similar to GFX_PixelBufferOffsetGet but performs no bounds checking.

	GFX_PixelBufferSet	Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer.
	GFX_PixelBufferSet_Unsafe	Sets a pixel in a pixel buffer at a point to a specified color. Caller is responsible for ensuring that the input color is in the same color format as the pixel buffer. Like GFX_PixelBufferSet but performs no bounds checking.

Structures

	Name	Description
	GFX_PixelBuffer_t	A pixel buffer is a wrapper around a basic data pointer. A pixel buffer has a color mode, a pixel count, a rectangular dimension, a pixel count, and a lenght in bytes.
	GFX_PixelBuffer	A pixel buffer is a wrapper around a basic data pointer. A pixel buffer has a color mode, a pixel count, a rectangular dimension, a pixel count, and a lenght in bytes.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Pixel buffer generation and management functions.

File Name

gfx_pixel_buffer.h

Company

Microchip Technology Inc.

gfx_rect.h

Defines general purposes rectangle functions.

Functions

	Name	Description
	GFX_RectClip	Clips a rectangle to the space of another rectangle. The result rectangle is a rectangle that will fit inside both of the given rectangles.
	GFX_RectClipAdj	This is function GFX_RectClipAdj.
	GFX_RectCombine	Combines the area of two rectangles into a single rectangle.
	GFX_RectCompare	This is function GFX_RectCompare.
	GFX_RectContainsPoint	Determines if a point is inside a rectangle.
	GFX_RectContainsRect	Determines if a rectangle is completely inside another rectangle. Still returns true if the edges are touching.
	GFX_RectFromPoints	This is function GFX_RectFromPoints.
	GFX_RectIntersects	Determines if two rectangles are intersecting
	GFX_RectsAreSimilar	This is function GFX_RectsAreSimilar.
	GFX_RectSplit	This is function GFX_RectSplit.
	GFX_RectToPoints	This is function GFX_RectToPoints.

Variables

	Name	Description
	GFX_Rect_Zero	This is variable GFX_Rect_Zero.

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Rectangle management functions.

File Name

gfx_rect.h

Company

Microchip Technology Inc.

gfx_util.h

Utility functions for the Hardware Abstraction Layer

Functions

	Name	Description
	GFX_UtilMirrorPoint	Reorients a point to a given mirrored orientation.
	GFX_UtilOrientPoint	Reorients a point to a given orthogonal orientation.
	GFX_UtilPointFromOrientedSpace	Transforms a point from an oriented rectangle space to an outer space given a display orientation and a mirroring setting.
	GFX_UtilPointToOrientedSpace	Transforms a point to an oriented rectangle space to an outer space given a display orientation and a mirroring setting.
	GFX_UtilSizeFromOrientedSpace	Transforms a size tuple from oriented space to screen space
	GFX_UtilSizeToOrientedSpace	Transforms a size tuple from screen space to oriented space
	GFX_UtilSortPointsX	Sorts two points in the X axis
	GFX_UtilSortPointsY	Sorts two points in the Y axis

Description

Module for Microchip Graphics Library - Hardware Abstraction Layer

Layer and point utility functions.

File Name

gfx_util.h

Company

Microchip Technology Inc.

Aria HAL Driver Examples

ILI9488 Display Controller Driver Library

Provides information on the ILI9488 Display Controller Driver Library

Description

This driver library provides an API interface to configure and use an external LCD module with an ILI9488 controller on a Microchip MCU. The driver is designed to work with the MPLAB Harmony Graphics Library.

Introduction

Introduces the driver library.

Description

The ILI9488 Display Controller Driver is a example implementation of a MPLAB Harmony Aria HAL driver, which supports DBI Type C 3-Line Serial Interface for MCUs with SPI peripheral, or DBI Type B 16-/8-bit parallel interface for MCUs with a Static Memory Controller (SMC) peripheral.

On these interfaces, the driver library provides APIs to:

- Send write and read commands to configure the ILI9488 Display Controller
- Write and read pixel(s) in the ILI9488's Graphics RAM (GRAM)

Using the Library

This topic describes the basic architecture of the ILI9488 Display Controller Driver Library and provides information and examples on how to use it.

Description

Interface Header File: `drv_gfx_ili9488.h`

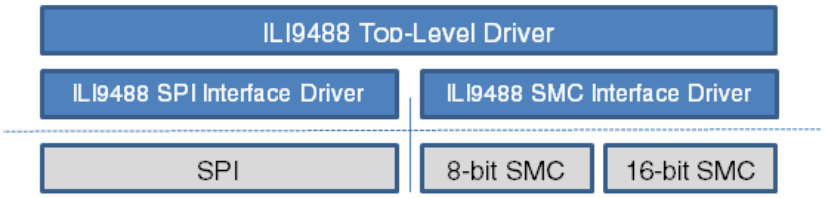
The top-level interface to the ILI9488 Display Controller Driver Library is defined in the `drv_gfx_ili9488.h` header file. Any C language source (`.c`), particularly in the graphics framework, file that uses the ILI9488 Display Controller Driver Library should include `drv_gfx_ili9488.h`.

Abstraction Model

This section describes how the ILI9488 Display Controller Driver Library provides the APIs that abstracts the interface-specific implementation from the application layer.

Description

The following figure shows a high-level block diagram that describes the structure of the display driver.



The top-level driver does the following:

- Glues the interface driver to the MPLAB Harmony Graphics Library's hardware abstraction layer
- Contains the ILI9488 setup routines and initialization commands, and calls to the interface driver layer
- Manages the buffer that contains pixel data that is sent to the ILI9488 Controller for display

The interface drivers provide the basic APIs for sending read/write commands, and reading/writing pixel data. The APIs contain corresponding calls to the interface's peripheral drivers (SPI or SMC) for communicating with the ILI9488 Controller.

LIbrary Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ILI9488 Display Controller Driver.

Library Interface Section	Description
Top-level Driver Functions	Functions that glues to the MPLAB Harmony graphics library and contains APIs for the graphics library to draw pixels to the ILI9488 GRAM.
Interface Driver Functions	Interface-specific APIs for sending write/read commands and pixel data to the ILI9488. These functions perform the necessary peripheral calls based on the selected interface (SPI or SMC).

How the Library Works

This section describes how the ILI9488 Display Controller Driver Library works .

Description

The library provides interfaces to support

- Initialization and setup of the ILI9488 Display Controller
- Sending read/write commands to the ILI9488 Display Controller using either the DBI-B parallel or DBI-C 4-Line SPI interface
- Reading/writing pixel data in the ILI9488 graphics RAM

Initialization

Before the driver can be used, it needs to be initialized. In ILI9488_ContextInitialize, the top-level driver registers itself to the MPLAB Harmony Graphics HAL, so that its initialization routine is automatically called when the graphics library initializes.

The initialization process involves allocating the pixel buffer and opening the interface to the ILI9488 Display Controller module. The top-level driver calls ILI9488_Intf_Open to open the peripheral interface, issues a hardware reset and sends the setup commands to the ILI9488. initCmdParm contains a list of the basic commands needed to set up the ILI9488 and these commands are sent through the interface driver using the ILI9488_Intf_WriteCmd API.

Displaying Frames

The top-level driver can be configured to write pixel data to the ILI9488 GRAM per pixel or per line. Per frame writes is also supported, but only in 16-bit DBI-B parallel mode. Per frame writes provides the fastest update rate, but requires more memory to allocate a full frame buffer. Writing per line has less memory space requirements, but requires an overhead to read the current pixel line data before it is updated and written back. This is done because the graphics library may update individual pixels only and not the whole line, thus the rest of the pixel buffer data must contain valid data before it is written back to the ILI9488 GRAM. Otherwise, the pixel buffer will contain pixel data from other lines and cause display artifacts.

To read and write pixel data, the top-level driver calls ILI9488_Intf_ReadPixels and ILI9488_Intf_WritePixels, respectively.

Library Interface

a) Functions

	Name	Description
	ILI9488_Intf_Close	Closes the HW interface to the ILI9488 device
	ILI9488_Intf_Open	Opens the specified port to the ILI9488 device
	ILI9488_Intf_ReadCmd	Sends read command and reads response from ILI9488
	ILI9488_Intf_ReadPixels	Read pixel data from specified position in ILI9488 GRAM
	ILI9488_Intf_WriteCmd	Sends write command and parameters to the ILI9488 device
	ILI9488_Intf_WritePixels	Writes pixel data to ILI9488 GRAM from specified position

b) Data Types and Constants

	Name	Description
	ILI9488_CMD_PARAM	Structure contains command and parameter information
	ILI9488_DRV_STATE	Enum of ILI9488 driver states
	ILI9488_DRV	Structure contains driver-specific data and ops pointers
	ILI9488_Backlight_Off	This is macro ILI9488_Backlight_Off.
	ILI9488_Backlight_On	This is macro ILI9488_Backlight_On.
	ILI9488_Reset_Assert	This is macro ILI9488_Reset_Assert.
	ILI9488_Reset_Deassert	This is macro ILI9488_Reset_Deassert.
	ILI9488_SPI_DCX_Command	This is macro ILI9488_SPI_DCX_Command.
	ILI9488_SPI_DCX_Data	This is macro ILI9488_SPI_DCX_Data.
	ILI9488_SPI_SS_Assert	This is macro ILI9488_SPI_SS_Assert.
	ILI9488_SPI_SS_Deassert	This is macro ILI9488_SPI_SS_Deassert.

Description

The interface described in the following sections are provided as an example of the Aria User Library Interface Hardware Abstraction Layer. This example can be used as a starting point for developing your own Aria HAL driver.

a) Functions

ILI9488_Intf_Close Function

Closes the HW interface to the ILI9488 device

File

help_drv_gfx_ili9488_common.h

C

```
void ILI9488_Intf_Close(ILI9488_DRV * drv);
```

Returns

None.

Description

This function will close the specified interface, free the port-specific data structures and unset the port operation handler functions.

Parameters

Parameters	Description
drv	ILI9488 driver handle

Function

void ILI9488_Intf_Close(ILI9488_DRV *drv)

ILI9488_Intf_Open Function

Opens the specified port to the ILI9488 device

File

help_drv_gfx_ili9488_common.h

C

```
GFX_Result ILI9488_Intf_Open(ILI9488_DRV * drv, unsigned int index);
```

Returns

- GFX_SUCCESS - Operation successful
- GFX_FAILURE - Operation failed

Description

In SPI mode, this function will open the SPI port, allocate the port-specific data structures and set the port operation handler functions. When done using the port, ILI9488_Intf_Close must be called to free up the data structures and close the port.

Parameters

Parameters	Description
drv	ILI9488 driver handle
index	Port index

Function

GFX_Result ILI9488_Intf_Open(ILI9488_DRV *drv, unsigned int index)

ILI9488_Intf_ReadCmd Function

Sends read command and reads response from ILI9488

File

help_drv_gfx_ili9488_common.h

C

```
GFX_Result ILI9488_Intf_ReadCmd(struct ILI9488_DRV * drv, uint8_t cmd, uint8_t * data, int bytes);
```

Returns

- GFX_SUCCESS Operation successful
- GFX_FAILURE Operation failed

Description

This function will fist write the the read command and then read back the response from the ILI9488 GRAM.

Remarks

This function only supports 8-, 24- or 32-bit reads. This function performs multiple full-blocking write/read calls to the SPI port and won't return until the SPI transaction completes.

Parameters

Parameters	Description
drv	ILI9488 driver handle
cmd	Read command
data	Buffer to store the read data to
bytes	Number of bytes to read

Function

GFX_Result ILI9488_Intf_ReadCmd(struct ILI9488_DRV *drv,

uint8_t cmd,

uint8_t *data,

int bytes);

ILI9488_Intf_ReadPixels Function

Read pixel data from specified position in ILI9488 GRAM

File

help_drv_gfx_ili9488_common.h

C

```
GFX_Result ILI9488_Intf_ReadPixels(struct ILI9488_DRV * drv, uint32_t x, uint32_t y, uint8_t * value,
unsigned int num_pixels);
```

Returns

- GFX_SUCCESS - Operation successful
- GFX_FAILURE - Operation failed

Description

This function will first write the start column, page information, then read the pixel data from the ILI9488 GRAM.

Remarks

For SPI mode, this function performs multiple full-blocking write/read calls to the SPI port and won't return until the SPI transaction completes.

Parameters

Parameters	Description
drv	ILI9488 driver handle
x	Column position
y	Page position
value	Value to store the read pixel color (8-bit/pixel RGB)
num_pixels	Number of pixels to read

Function

GFX_Result ILI9488_Intf_ReadPixels(struct ILI9488_DRV *drv,
uint32_t x,
uint32_t y,
uint16_t *value,
unsigned int num_pixels)

ILI9488_Intf_WriteCmd Function

Sends write command and parameters to the ILI9488 device

File

help_drv_gfx_ili9488_common.h

C

```
GFX_Result ILI9488_Intf_WriteCmd(struct ILI9488_DRV * drv, uint8_t cmd, uint8_t * parms, int num_parms);
```

Returns

- GFX_SUCCESS - Operation successful
- GFX_FAILURE - Operation failed

Description

This function will do a write operation to send the write command and its parameters to the ILI9488.

Remarks

In SPI mode, this is a full-blocking call and waits for the SPI transaction to complete.

Parameters

Parameters	Description
drv	ILI9488 driver handle
cmd	Write command
parms	Pointer to array of 8-bit parameters
bytes	number of command parameters

Function

GFX_Result ILI9488_Intf_WriteCmd(struct ILI9488_DRV *drv,
uint8_t cmd,
uint8_t *parms,
int num_parms)

ILI9488_Intf_WritePixels Function

Writes pixel data to ILI9488 GRAM from specified position

File

help_drv_gfx_ili9488_common.h

C

```
GFX_Result ILI9488_Intf_WritePixels(struct ILI9488_DRV * drv, uint32_t start_x, uint32_t start_y, uint8_t *
data, unsigned int num_pixels);
```

Returns

- GFX_SUCCESS - Operation successful
- GFX_FAILURE - Operation failed

Description

This function will fist write the start column, page information, then write the pixel data to the ILI9488 GRAM.

Remarks

In SPI mode, this function performs multiple full-blocking write calls to the SPI port and won't return until the SPI transaction completes.

Parameters

Parameters	Description
drv	ILI9488 driver handle
start_x	Start column position
start_y	Start page position
data	Array of 8-bit pixel data (8-bit/pixel RGB)
num_pixels	Number of pixels

Function

GFX_Result ILI9488_Intf_WritePixels(struct ILI9488_DRV *drv,

uint32_t start_x,

uint32_t start_y,

uint8_t *data,

unsigned int num_pixels)

b) Data Types and Constants

ILI9488_CMD_PARAM Structure

Structure contains command and parameter information

File

help_drv_gfx_ili9488_common.h

C

```
typedef struct {
  uint8_t cmd;
  uint8_t parmCount;
  uint8_t parms[4];
} ILI9488_CMD_PARAM;
```

Members

Members	Description
uint8_t cmd;	Command
uint8_t parmCount;	Number of command parameters
uint8_t parms[4];	Command parameters, max of 4

Description

- ILI9488_CMD_PARAM

ILI9488_DRV_STATE Enumeration

Enum of ILI9488 driver states

File

help_drv_gfx_ili9488_common.h

C

```
typedef enum {
  INIT = 0,
  RUN
} ILI9488_DRV_STATE;
```

Description

- ILI9488_DRV_STATE

ILI9488_DRV Structure

Structure contains driver-specific data and ops pointers

File

help_drv_gfx_ili9488_common.h

C

```
struct ILI9488_DRV {
  GFX_Context* gfx;
  ILI9488_DRV_STATE state;
  uint8_t * pixelBuffer;
  int currentLine;
  int lineX_Start;
  int lineX_End;
  GFX_Bool linePending;
  unsigned int bytesPerPixelBuffer;
  void * port_priv;
};
```

Members

Members	Description
GFX_Context* gfx;	GFX context pointer
ILI9488_DRV_STATE state;	Driver state
uint8_t * pixelBuffer;	Line buffer information
unsigned int bytesPerPixelBuffer;	bytes per pixel buffer
void * port_priv;	Port-specific private data

Description

- ILI9488_DRV

ILI9488_Backlight_Off Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_Backlight_Off BSP_DisplayBacklightStateSet(0)
```

Description

This is macro ILI9488_Backlight_Off.

ILI9488_Backlight_On Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_Backlight_On BSP_DisplayBacklightStateSet(1)
```

Description

This is macro ILI9488_Backlight_On.

ILI9488_Reset_Assert Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_Reset_Assert BSP_DisplayResetStateSet(0)
```

Description

This is macro ILI9488_Reset_Assert.

ILI9488_Reset_Deassert Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_Reset_Deassert BSP_DisplayResetStateSet(1)
```

Description

This is macro ILI9488_Reset_Deassert.

ILI9488_SPI_DCX_Command Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_SPI_DCX_Command BSP_ILI9488_SPI_DCXStateSet(0)
```

Description

This is macro ILI9488_SPI_DCX_Command.

ILI9488_SPI_DCX_Data Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_SPI_DCX_Data BSP_ILI9488_SPI_DCXStateSet(1)
```

Description

This is macro ILI9488_SPI_DCX_Data.

ILI9488_SPI_SS_Assert Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_SPI_SS_Assert BSP_ILI9488_SPI_CSXStateSet(0)
```

Description

This is macro ILI9488_SPI_SS_Assert.

ILI9488_SPI_SS_Deassert Macro

File

help_drv_gfx_ili9488_common.h

C

```
#define ILI9488_SPI_SS_Deassert BSP_ILI9488_SPI_CSXStateSet(1)
```

Description

This is macro ILI9488_SPI_SS_Deassert.

Nano2D Graphics Processing Unit (GPU) Driver Library

Provides information on the Nano 2-Dimensional (Nano2D) Driver Library.

Description

Nano 2-Dimensional (Nano2D) is a driver library for rendering 2-Dimensional computer graphics. It is the means for hardware-accelerated graphics on PIC32MZ microcontrollers containing the 2-Dimensional Graphics Processing Unit (2-D GPU).

Introduction

This topic provides the introduction to the Nano2D Driver Library.

Description

The Nano2D Library API provides full functionality of the PIC32MZ 2-D GPU module, which includes lines, rectangles, bit block transfers (blits), transparency, and binary Raster Operations (ROP2). These features are employed in GFX application demonstrations compiled with Nano2D on 2-D GPU enabled microcontrollers. Blits can be used to quickly transfer pre-rendered images directly to the display's frame buffer. (Pre-rendering images converts image pixels from JPEG or some other format into raw RGB or RGBA bits.)

In Harmony 2.06 there is a new tool, the DDR Organizer, that assists in managing buffers, raw images, and other memory resources in the DDR memory of DA devices. Images can quickly and easily be organized for pre-rendering at boot-up so that later image draws will use the device's Nano2D GPU.

The library provides low-level 2D primitives while containing no facilities for GUI development, therefore, it can be used as a stand-alone API. If GUI development is desired, it can be achieved using a higher level widget library such as, libAria which is embedded in the MPLAB Harmony Graphics Suite.

Note: Nano2D requires MPLAB Harmony v2.02b or later.

MPLAB Harmony Graphics Suite has two memory rendering options:

- Aria – Uses Hardware-independent software rasterizer – software fallback
- Nano2D – Uses 2-D GPU hardware accelerated peripheral

In MPLAB Harmony v2.03 and later, the MPLAB Harmony Graphics Suite uses Nano2D as its default memory buffer rendering interface when developing for PIC32 2D-GPU enabled devices. This default can be overridden in MHC by user deselection of Nano2D Graphics Processor.

Nano2D provides the API for drawing accelerated raster graphics onto memory buffers. The actual drawing happens in the 2-D GPU peripheral. Nano2D can be a better option over libAria's hardware-independent graphics primitives because Nano2D uses little to no CPU resources. The use of the Nano2D library requires that the DA's built-in 2D graphics processor be enabled. Under *Harmony Framework Configuration* > *Graphics Stack* > *Graphics Processor*, select the NANO 2D processor.

Using the Library

This topic describes how to use the Nano2D Driver Library.

Description

Interface Header File: `libnano2d.h`

The interface to the Nano2D Library is defined in the `libnano2d.h` header file. Any C language source (`.c`) file that uses the Nano2D Library should include `libnano2d.h`. The header file can be found within the following MPLAB Harmony directory.

`<install_dir>/framework/gfx/driver/processor/nano2d/libnano2D.a`

Library File: `libnano2d.a`

The Nano2D Library archive (libnano2d.a) file is installed with MPLAB Harmony. The header file can be found within the following MPLAB Harmony directory.

`<install_dir>/framework/bin/driver/processor/nano2d/libnano2D.a`

Please refer to What is MPLAB Harmony? for how the Nano2D Library interacts with the framework.

`<install_dir>/framework/gfx/driver/processor/nano2d/libnano2D.h`

Abstraction Model

Provides information on the abstraction model for the library.

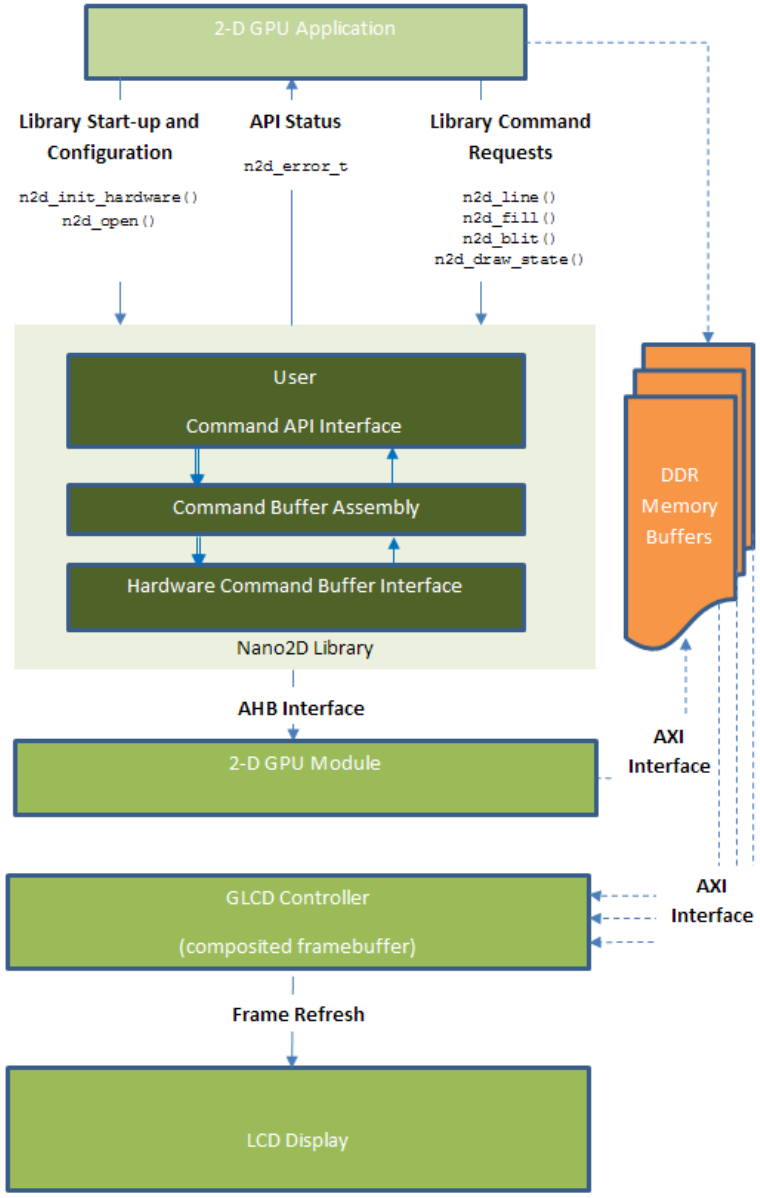
Description

The Nano2D Driver Library is the single interface to the 2-D GPU Module. The 2-D GPU does not have Special Function Register (SFR) access like many PIC32 peripherals. Alternatively, Nano2D uses a command buffer, shared between it and the 2-D GPU, to communicate request. Nano2D Driver Library builds the command buffer content based on each API request and makes it available at a physical address known to the GPU. The address of the command buffer address location is established at startup before any request arrives.

The communication between application and Nano2D is C synchronous function calls. Each function returns on completion of requested. This is typically between 0 -1ms. The communication between Nano2D and 2-GPU is the command buffer and internal well-known status registers. These registers are not exposed to the application.

The Nano2D Driver Library commits a complex command buffer protocol for each GPU request. The Nano2D Driver Library removes the overhead of command buffer assembly and status from the application. This leaves the application with an easy to use, synchronous, return code interface.

Nano2D Software Abstraction Block Diagram



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

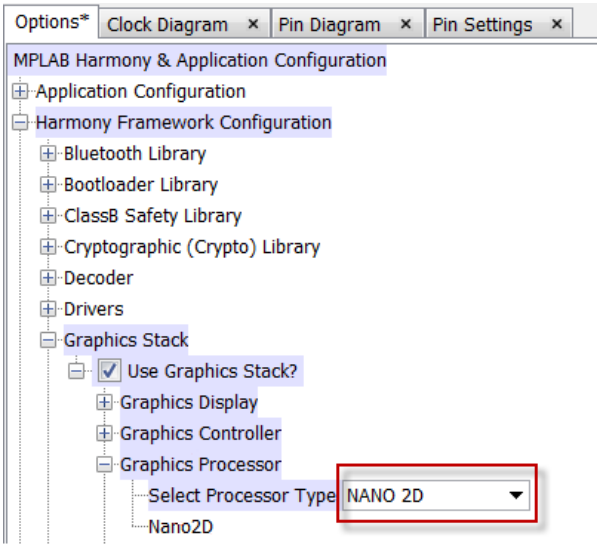
The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Nano2D Driver.

How the Library Works

This topic provides information on how the library works.

Description

The Nano2D Driver Library service provides general APIs for graphics application use. To enable Nano2D, the user is required to select the Nano2D as the graphics processor using MHC within the “Options*” tab. Upon generation, the `libnano2d.a` library, Nano2D initialization code, and header file will be added to the project. There is no additional configuration required.



The following app.c code example shows typical usage of this Nano2D Library.
[Code Example]

```
void APP_Tasks ( void )
{
/* Check the application's current state. */
switch ( appData.state )
{
/* Application's initial state. */
case APP_STATE_INIT:
{
bool appInitialized = true;

if (appInitialized)
{
appData.state = APP_STATE_GPU_FILL;
}
break;
}

case APP_STATE_GPU_FILL:
{
n2d_rectangle_t rect;
n2d_color_t redColor = 0x00ff000000;
n2d_color_t greenColor = 0x0000ff0000;

/* create GPU buffer for GLCD layer 0 (dest) */
layer0.format = N2D_RGBA8888;
layer0.gpu = KVA_TO_PA(0xA8000000); //GLCD display layer 0
layer0.memory = (void*)0xA8000000;
layer0.width = 480; // width of buffer same as display width
layer0.height = 272; // height of buffer same as display height
layer0.orientation = N2D_0;
layer0.stride = layer0.width * 32 / 8;

/* fill entire buffer 480x272 */
n2d_fill(layer0, N2D_NULL, redColor, N2D_BLEND_NONE);

/* create a clipping rectangle for layer0 */
rect.x = 0; rect.y = 0; rect.width = 100; rect.height = 100;

/* fill 100x100 rectangle with green ADDITIVE blend to
make color brown */
n2d_fill(layer0, &rect, greenColor, N2D_BLEND_ADDITIVE);

appData.state = APP_STATE_IDLE;

}
}
```

Drawing

Nano2D provides basic drawing primitives: n2n_line and n2n_rect. These functions support integer only coordinates. If more complex graphics is required, the user will need to use a higher level graphics library, such as Aria Graphics Library which is one part of the MPLAB Harmony Graphics Suite. Aria can be configured to call Nano2D as its 2D primitive memory interface renderer for acceleration.

The Line operation, n2n_line, draws a line. Two points are given: start point and end point. The end point is not drawn for point to point figure drawing is required. Lines are rendered using the Breshenham algorithm. Clipping is supported for lines on a per pixel basis. To draw a line, use the following statement:

```
n2d_line(destination, start, end, clip, color, blend);
```

The Rectangle operation, n2n_rect, draws a rectangle. It fills or draws a rectangle area with a given color. A rectangle is given the top left coordination and bottom right coordinate of the fill region, the fill color, clipping region, and blending mode which is applied to each pixel. To draw a rectangle, use the following statement:

```
n2d_fill(destination, rectangle, color, blend);
```

There are 7 types of blend modes:

- N2D_BLEND_NONE - S, i.e. no blending
- N2D_BLEND_SRC_OVER - S + (1 - Sa) * D
- N2D_BLEND_DST_OVER - (1 - Da) * S + D
- N2D_BLEND_SRC_IN - Da * S
- N2D_BLEND_DST_IN - Sa * D
- N2D_BLEND_ADDITIVE - S + D
- N2D_BLEND_SUBTRACT - D * (1 - S)

Where, Sa and Da represent the source and destination alpha channels.

Block Transfers of Pixels (Blitting)

Nano2D provides the following blit operations: Blit, Stretch/Shrink Blit, Mask, Blit, and automatic Filter blit during stretch and shrink.

The Blit operation transfers data from one area of a memory source to another area of a memory destination. The source and destination can be from the same or from different memory locations. Both source and destination must be described by a rectangle. Blitting supports automatic behavior:

- Stretch/Shrink – If source and destination rectangles are different sizes the operation becomes a stretch or a shrink blit
- Mask – use of ROPs for transparent pixels
- Monochrome – using ROPs for monochrome images

Stretch blit is not allowed to overlap, that is no part of the source and destination can share the same portion of memory. Non-stretch blits can overlap.

Blits supports the 7 blending modes discussed previously which are applied to each pixel.

To blit, use the following statement:

```
n2d_blit(dst, dst_rect, src, src_rect, blend);
```

Transparency

Nano2D provides a means of controlling transparency applied to each pixel for subsequent draw commands. Transparency is also synonymous to masking and operation. Nano2D uses Binary Raster Operations (ROP2) to affect action on each pixel during blitting. The user can set a transparency mode or turn off transparency.

The Draw State operation, n2n_draw_state, sets the drawing operation for subsequent n2d_blit() calls.

To change draw state, use the following statement:

```
n2d_draw_state (transparency, color, src, foreground_rop, background_rop);
```

The statement is executed like the C condition “?” ternary operator that takes three values. It reads: If color in transparency_mode is true then perform foreground_rop on matching color otherwise perform background_rop on all other non-matching colors. Color equates to a pixel.

The following standard Binary ROPs are supported:

ROP	Formula	Description
0x0	`0`	Set all destination bits to 0.
0x1	`~(D|S)`	Inverse of merge source and destination.
0x2	`D&~S`	Inverse of merge source and destination.
0x3	`~S`	Inverse of merge source and destination.
0x4	`S&~D`	Mask source and inverse of destination.
0x5	`~D`	Invert destination.
0x6	`D^S`	Exclusive or of source and destination.
0x7	`~(D&S)`	Inverse of mask source and destination.
0x8	`D&S`	Inverse of mask source and destination.

0x9	~(D^S)	Inverse of mask source and destination.
0xA	D	Copy destination.
0xB	D|~S	Merge inverse of source and destination.
0xC	S	Copy source.
0xD	S|~D	Merge source and inverse of destination.
0xE	D|S	Merge source and destination.
0xF	1	Set all destination bits to 1.

Clipping

Nano2D provides a clipping rectangle for line, rectangles and blits. Clipping is performed on a per pixel basis. For all functions, the clipping area is defined by n2d_rectangle_t.

Memory Buffers

Nano2D provides an abstraction over a memory buffer region. Each function defined in `libnano2D.h` uses the n2d_buffer_t to establish the portion of shared memory used as a source or destination buffer. The structure contains all the information the libnano2D APIs is required to complete a GPU render command request. See "Data Types and Constants" in the Library Interface section.

The Nano2D library supports four buffer and their alpha swizzle formats. They are: RGBA8888, RGB565, RGB4444, and A8. When using Nano2D through Aria Library, only supports the two major RGB formats RGB8888 and RGB565.

RGBA8888 (24bit true-color) memory buffer establishes green blue red channels for color space with an extra alpha channel for color blending. Requiring 32bits per-pixel, it will consume a large buffer space to hold display content.

RGB565 (reduced color) memory buffer uses half the color space, has no alpha channel, and, with CPU processing, proves to be a higher performer than RGBA8888.

A8 is an all-alpha buffer. It is can be used to produced a greyscale (dimming) of an existing RBGA8888 buffer.

The buffer supports the following:

- `width` - Width of the buffer in pixels
- `height` - Height of the buffer in pixels
- `stride` - Stride of the buffer in bytes
- `format` - Pixel format of the buffer
- `orientation` - Buffer's orientation: 0, 90, 180, 270
- `memory` - Logical pointer to the buffer's memory for the CPU
- `gpu` - Physical address of the buffer's memory the hardware can access

Detailed Use

The 2-D GPU is made available through the Nano2D Library Module. In future releases, MHGC will be its higher level access. The remainder of these sections will describe how to command the 2-D GPU directly without using MHGC using customized C code.

Unlike most controller peripherals, 2-D GPU command register is not available to application developers. There are no Special Function Registers (SFRs). As a replacement to SFR access, the Nano2D Library provides a C interface API implemented within the libnano2D.a object file. When linked to the application, these entities provide command access to 2-D GPU.

Nano2D Library and 2-D GPU communicate through a shared buffer region residing in physical memory. This memory location is provided by default in Harmony. For rendering, Nano2D uses user supplied source and destination buffers. These buffers communicate the necessary details of the memory region which includes size, color depth, location, and orientation. One or more of these buffers are required for 2-D GPU rendering APIs using `libnano2D.a`.

Creating a Pixel Buffer

An n2d_buffer_t data structure maintains the context of the rendering memory buffer. A memory buffer can point to a scratch buffer or to the active framebuffer. In these examples, we will draw to the active framebuffer as well as scratch memory buffers.

The characteristics of the currently displayed framebuffer must be understood. These include the physical address of the framebuffer, its size, color depth, and orientation. An n2d_buffer_t structure must be created to contain this information. If the n2d_buffer_t is different than the framebuffer, rendered graphics will have unexpected behavior. After generation, the user will need to view the generated `system_config.h` file and `libnano2d.h`. Use the following steps to create an n2d_buffer_t structure that points to the active framebuffer.

```
n2d_buffer_t layer0;

layer0.width = GFX_GLCD_LAYER0_RES_X; // see system_config.h
layer0.height = GFX_GLCD_LAYER0_RES_Y;
layer0.stride = layer0.width * 32 /8; // 32bits/8 == 4 bytes
layer0.format = N2D_RGBA8888; // Red Green Blue Alpha 32bbp
layer0.orientation = N2D_0; // 0 degree orientation
layer0.handle = GFX_NULL; // handle is unused
layer0.memory = (void*)GFX_GLCD_LAYER0_BASEADDR;
```

```
layer0.gpu = KVA_TO_PA(GFX_GLCD_LAYER0_BASEADDR);
```

`layer0.gpu` is the starting location in DDR memory from which 2-D GPU will read or write. This is a physical memory address. If the application must modify data at this location, it must use layer0.memory which maintains the virtual (accessible) address.

Drawing a Grid of Lines

Lines are primitive graphic items necessary to higher level widgets. To render a line directly to the framebuffer, the user will need to know the location of the framebuffer, the start and end of the line, the color, and blend factors of the line.

Note: The color parameter is in ARGB format and is not aligned with the buffer format. The following code example demonstrates line drawing by creating a perpendicular grid.

```
n2d_buffer_t * buffer = layer0;
n2d_point_t start, end;
n2d_color_t color;
n2d_int32_t i;

color = 0xff00ffff;

n2d_fill(buffer, N2D_NULL, color, N2D_BLEND_NONE);

/* Draw vertical line. */
start.x = 0;
start.y = 5;
end.x = 0;
end.y = buffer->height - 5;

color = 0xffff0000;

for (i = 0; i < buffer->width / 10; i++)
{
n2d_line(buffer, start, end, N2D_NULL, color, N2D_BLEND_NONE);

start.x += 10;
end.x += 10;
}

/* Draw horizontal line. */
start.x = 5;
start.y = 0;
end.x = buffer->width - 5;
end.y = 0;

for (i = 0; i < buffer->height / 10; i++)
{
n2d_line(buffer, start, end, N2D_NULL, color, N2D_BLEND_NONE);

start.y += 10;
end.y += 10;
}
```

Drawing Cascading Blended Rectangles

Rectangles are primitive graphic items necessary to higher level widgets. The 2-D GPU can render these quickly with alpha-blending and orientation settings. To render cascading rectangles directly to the framebuffer, the user will need to know the location of the framebuffer, the top, left, width, height (rect), the color, and blend factors of the rect. The following code example integrates rectangle drawing by creating a rectangles in a cascading order using different colors.

```
n2d_buffer_t * buffer = layer0;
n2d_point_t start, end;
n2d_color_t color;
n2d_int32_t i;

/* Clear background color to black. */
color = 0xff000000;
n2d_fill(buffer, N2D_NULL, color, N2D_BLEND_NONE);

rect.x = 0;
rect.y = 0;
rect.width = buffer->width / 4;
```

```
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0xffff0000, N2D_BLEND_SRC_OVER);

rect.x = buffer->width / 8;
rect.y = buffer->height / 8;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;

n2d_fill(buffer, &rect, 0x7f00ff00, N2D_BLEND_SRC_OVER);

rect.x = buffer->width / 4;
rect.y = buffer->height / 4;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0xffffffff, N2D_BLEND_SRC_OVER);

rect.x = buffer->width * 3 / 8;
rect.y = buffer->height * 3 / 8;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0x700000ff, N2D_BLEND_SRC_OVER);

rect.x = buffer->width * 4 / 8;
rect.y = buffer->height * 4 / 8;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0xffffff00, N2D_BLEND_SRC_OVER);

rect.x = buffer->width * 5 / 8;
rect.y = buffer->height * 5 / 8;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0x7fff00ff, N2D_BLEND_SRC_OVER);

rect.x = buffer->width * 6 / 8;
rect.y = buffer->height * 6 / 8;
rect.width = buffer->width / 4;
rect.height = buffer->height / 4;
n2d_fill(buffer, &rect, 0xff00ffff, N2D_BLEND_SRC_OVER);
```

Image Source and Destination Rotation

A copy of one buffer source region to another is called blitting. The example below, blits an image rendered to a source buffer onto a destination buffer. The destination buffer points to the framebuffer. For this example, a staging src buffer to hold the image is established. Also a small set of GFX Hal APIs are used to decode the image to the src buffer.

```
n2d_buffer_t src0;
src0.format = N2D_RGBA8888;
src0.gpu = KVA_TO_PA(0xA8465000);
src0.memory = (void*)0xA8465000;
src0.width = 256;
src0.height = 256;
src0.orientation = N2D_0;
src0.stride = src0.width * 32 / 8;

/* use gfx hal to render image to source buffer 0 */
GFX_PixelBufferCreate(256,
256,
GFX_COLOR_MODE_RGBA_8888,
(uint32_t*)src0.memory,
&pixelBuffer);

GFX_Begin();
GFX_Set(GFXF_DRAW_TARGET, &pixelBuffer);
GFXU_DrawImage(&image0,
0,
0,
256,
256,
0,
```

```
0,
NULL,
NULL);
GFX_Set(GFXF_DRAW_TARGET, NULL);
GFX_End();

/* use gfx hal to render image to source buffer 1 */
GFX_PixelBufferCreate(256,
256,
GFX_COLOR_MODE_RGBA_8888,
(uint32_t*)src1.memory,
&pixelBuffer);

GFX_Begin();
GFX_Set(GFXF_DRAW_TARGET, &pixelBuffer);
GFXU_DrawImage(&image1,
0,
0,
256,
256,
0,
0,
NULL,
NULL);
GFX_Set(GFXF_DRAW_TARGET, NULL);
GFX_End();
```

Raster Operations

The processing of source pixels onto destination pixels is called a raster operation. The 2-D GPU through the libnano2D library support two input (binary) Raster Operations (ROP2). These are industry standard bitwise logical operations defined by 16 possible functions listed above in the Transparency section.

In order to set the operation for subsequent draw functions, n2d_buffer_t is used. The function can be continuously used to set mask/filter operations or turn transparency off.

Its arguments are similar to C conditional “?” statements. The first argument determines the operation mode. The operation mode determines whether color is to be applied to N2D_TRANSPARENCY_NONE (no pixels) or N2D_TRANSPARENCY_SOURCE or N2D_TRANSPARENCY_DESTINATION buffer. If the color matches, then the foreground operation is applied, otherwise the background operation is applied.

Consider the following example statement:

```
n2d_draw_state(N2D_TRANSPARENCY_SOURCE, 0xff0000, 0xe, 0xc);
```

The statement will inform the GPU to look for the color red in the source the source buffer during n2d_blit. If the pixel is found, the final pixel will become a merge of the original source and destination pixels, otherwise the pixel will become a copy of the original source pixel.

The following code example applies all 16 ROP functions on green (source) and blue (destination) pixels.

```
n2d_buffer_t src, *buffer;
n2d_uint8_t rop = 0;
n2d_uint32_t x, y;
n2d_int32_t deltaX, deltaY;
n2d_rectangle_t rect;

buffer = &layer0;

deltaX = buffer->width >> 2;
deltaY = buffer->height >> 2;

/* Fill the source buffer with green color. */
n2d_fill(&src, N2D_NULL, 0xff00, N2D_BLEND_NONE);

/* Fill the dst buffer with blue color. */
n2d_fill(buffer, N2D_NULL, 0xff, N2D_BLEND_NONE);

/* Loop all rop values. */
for (y = 0; y < 4; y++)
{
for (x = 0; x < 4; x++)
{
rect.x = x * deltaX;
rect.y = y * deltaY;
rect.width = deltaX;
```

```
rect.height = deltaY;

/* Set rop. */
n2d_draw_state(N2D_TRANSPARENCY_NONE, 0x0, rop, rop);

n2d_blit(buffer, &rect, &src, &rect, N2D_BLEND_NONE);
rop++;
}
}

n2d_draw_state(N2D_TRANSPARENCY_NONE, 0x0, 0xc, 0xc);
Alpha Greyscale -<< Red Color "section"
/* alpha buffer for blending */
alpha.format = N2D_A8;
alpha.gpu = KVA_TO_PA(0xA85DC000);
alpha.memory = (void*)0xA85DC000;
alpha.width = appData.display_info->rect.width;
alpha.height = appData.display_info->rect.height;
alpha.orientation = orientation;
alpha.stride = alpha.width * 8 / 8;
/* dim pixels within a rectangle area */
/* Init the alpha buffer with an alpha channel. */
memset(alpha.memory, 0x07, alpha.stride * alpha.height);
/* Blit – subtract alpha value on all pixels of destination */
 n2d_blit(&layer0, N2D_NULL, &alpha, N2D_NULL, N2D_BLEND_SUBTRACT);
```

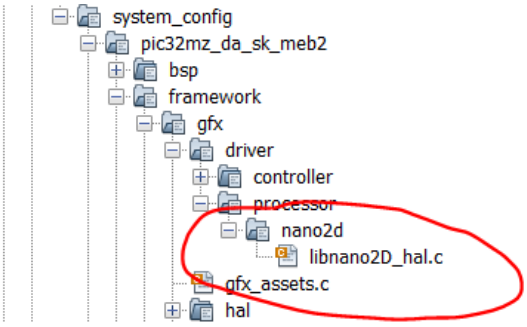
Configuring the Library

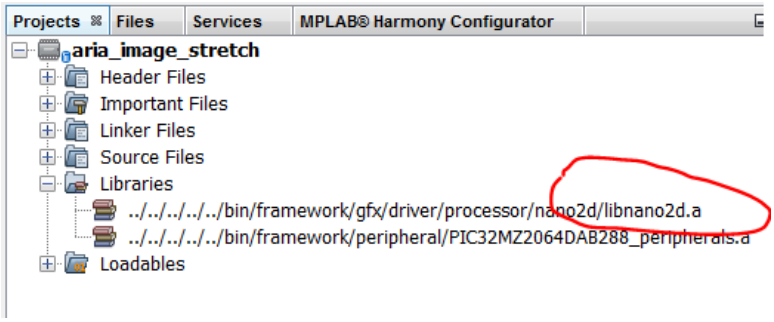
This topic describes how to configure the Nano2D Driver Library.

Description

The configuration of the Nano2D module is based on Nano2D processor selections of MHC. There are no additional configurations available for Nano2D. To enable Nano2D, the user is required to select the Nano2D as the graphics processor using MHC under “Options*”. Upon generation, the `libnano2d.a` library, Nano2D initialization code, and header file will be added to the project.

Upon generation, the `libnano2d.a` library, Nano2D initialization code, Nano2D thin adapter file, and header file will be added to the project.





Building the Library

This topic explains building the Nano2D Driver Library.

Description

The library is provided in binary form only, and comes prebuilt when you install MPLAB Harmony. In addition, the library is added to your project when Nano2D is selected and the project is generated.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/gfx/driver/processor/nano2d`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/libnano2D.h`	Header file that exports the driver API.

Library File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/libnano2D.a`	The static library that contains the implementation of `libnano2D.h`.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Nano2D Driver Library depends on the following modules:

- GLCD Graphics Display Controller (currently, the Nano2D module is initialized within the GLCD display controller)

Library Interface

This section describes the Application Programming Interface (API) functions of the Nano2D module.

a) Functions

	Name	Description
	n2d_blit	Copy a source buffer to the the destination buffer
	n2d_draw_state	Set the drawing state for any following Nano2D API draw call
	n2d_fill	Fill a (partial) buffer with a specified color
	n2d_init_hardware	Initializes the n2d driver and peripheral hardware
	n2d_line	Draw a line
	n2d_open	Open Nano2d context
	n2d_dither	Enable or disable dithering

b) Data Types and Constants

	Name	Description
	n2d_blend	List of blending modes
	n2d_buffer	A wrapper structure for any image or render target
	n2d_buffer_format	List of blending modes
	n2d_error	Error codes that the Nano2D functions can return
	n2d_module_parameters	GPU peripheral Initialization parameters
	n2d_orientation	List of blending modes
	n2d_point	A position on a pixel
	n2d_rectangle	A rectangle
	n2d_transparency	Transparency modes
	n2d_blend_t	List of blending modes

	n2d_buffer_format_t	List of blending modes
	n2d_buffer_t	A wrapper structure for any image or render target
	n2d_color_t	Identifies a specific pixel color
	n2d_error_t	Error codes that the Nano2D functions can return
	n2d_module_parameters_t	GPU peripheral Initialization parameters
	n2d_orientation_t	List of blending modes
	n2d_point_t	A position on a pixel
	n2d_rectangle_t	A rectangle
	n2d_transparency_t	Transparency modes
	n2d_bool_t	This is type n2d_bool_t.
	n2d_float_t	This is type n2d_float_t.
	n2d_int16_t	This is type n2d_int16_t.
	n2d_int32_t	This is type n2d_int32_t.
	n2d_size_t	This is type n2d_size_t.
	n2d_uint16_t	This is type n2d_uint16_t.
	n2d_uint32_t	This is type n2d_uint32_t.
	n2d_uint64_t	This is type n2d_uint64_t.
	n2d_uint8_t	This is type n2d_uint8_t.
	__gcmALIGN	This is macro __gcmALIGN.
	__gcmEND	This is macro __gcmEND.
	__gcmGETSIZE	This is macro __gcmGETSIZE.
	__gcmMASK	This is macro __gcmMASK.
	__gcmSTART	This is macro __gcmSTART.
	_nano2D_types_h__	This is macro _nano2D_types_h__.
	gcmALIGN	This is macro gcmALIGN.
	gcmCOUNTOF	This is macro gcmCOUNTOF.
	gcmGETFIELD	This is macro gcmGETFIELD.
	gcmINT2PTR	This is macro gcmINT2PTR.
	gcmMAX	This is macro gcmMAX.
	gcmMIN	This is macro gcmMIN.
	gcmPTR2INT	This is macro gcmPTR2INT.
	gcmSETFIELD	This is macro gcmSETFIELD.
	gcmSETFIELDVALUE	This is macro gcmSETFIELDVALUE.
	gcmSETMASKEDFIELD	This is macro gcmSETMASKEDFIELD.
	gcmSETMASKEDFIELDVALUE	This is macro gcmSETMASKEDFIELDVALUE.
	gcmVERIFYFIELDVALUE	This is macro gcmVERIFYFIELDVALUE.
	IN	This is macro IN.
	N2D_FALSE	This is macro N2D_FALSE.
	N2D_INFINITE	This is macro N2D_INFINITE.
	N2D_IS_ERROR	This is macro N2D_IS_ERROR.
	N2D_IS_SUCCESS	This is macro N2D_IS_SUCCESS.
	N2D_NULL	This is macro N2D_NULL.
	N2D_ON_ERROR	This is macro N2D_ON_ERROR.
	N2D_TRUE	This is macro N2D_TRUE.
	OUT	This is macro OUT.

Description

Refer to each section for a detailed description.

a) Functions

n2d_blit Function

Copy a source buffer to the the destination buffer

File

libnano2D.h

C

```
n2d_error_t n2d_blit(n2d_buffer_t * destination, n2d_rectangle_t * destination_rectangle, n2d_buffer_t *
source, n2d_rectangle_t * source_rectangle, n2d_blend_t blend);
```

Returns

Returns the status as defined by n2d_error_t

Description

The specified region of the source buffer is copied to the specified region of the destination buffer. If the regions are different in size, simple low-quality scaling will automatically be performed.

An optional blend mode can be specified that defines the blending of the source onto the destination.

Remarks

This function will wait until the hardware is complete, i.e. it is synchronous.

Parameters

Parameters	Description
destination	Pointer to a n2d_buffer_t structure that describes the destination of the blit
destination_rectangle	Optional pointer to the rectangle that defines the region inside the destination buffer. If this rectangle is not specified, the entire destination buffer is used as the destination region
source	Pointer to a n2d_buffer_t structure that describes the source of the blit
source_rectangle	Optional pointer to the rectangle that defines the region inside the source buffer. If this rectangle is not specified, the entire source buffer is used as the source region
blend	Optional blending mode to be applied to each pixel. If no blending is required, set this value to N2D_BLEND_NONE (0)

Function

n2d_error_t n2d_blit()

n2d_draw_state Function

Set the drawing state for any following Nano2D API draw call

File

libnano2D.h

C

```
n2d_error_t n2d_draw_state(n2d_transparency_t transparency, n2d_color_t color, n2d_uint8_t foreground_rop,
n2d_uint8_t background_rop);
```

Returns

Returns the status as defined by n2d_error_t

Description

In order to setup transparency for the n2d_blit function, this function needs to be called. Note that this function is static, so set once, all draw commands that follow this function will take this transparency into effect. Call this function again with different parameters to set another transparency mode or turn transparency off.

It will return N2D_INVALID_ARGUMENT if the source defines the transparency but the rop has nothing to do with the source buffer.

The default transparency mode for any newly opened context is N2D_TRANSPARENCY_NONE, using a foreground_rop of 0xC (copy source).

Binary ROPs supported in both foreground and background operations: ROP Formula Description 0x0 0 Set all destination bits to 0. 0x1 ~(D|S) Inverse of merge source and destination. 0x2 D&~S Mask inverse of source and destination. 0x3 ~S Copy inverse of source. 0x4 S&~D Mask source and inverse of destination. 0x5 ~D Invert destination. 0x6 D^S Exclusive or of source and destination. 0x7 ~(D&S) Inverse of mask source and destination. 0x8 D&S Mask source and destination. 0x9 ~(D^S) Inverse of exclusive or of source and destination. 0xA D Copy destination. 0xB D|~S Merge inverse of source and destination. 0xC S Copy source. 0xD S|~D Merge source and inverse of destination. 0xE D|S Merge source and destination. 0xF 1 Set all destination bits to 1.

Remarks

When using a source buffer with the A8 pixel format, transparency must be enabled to N2D_TRANSPARENCY_SOURCE and the alpha channel of color will be used to check for transparency. If the pixel is not transparent, the RGB channels of color value will be used as the color for the pixel.

Parameters

Parameters	Description
transparency	The transparency mode applied to each pixel. See n2d_transparency_t for a list of all supported transparency modes
color	If transparency is not N2D_TRANSPARENCY_NONE, this color value specifies if a pixel is a foreground or a background pixel. If the color matches, it is a background pixel, otherwise it is a foreground pixel
foreground_rop	A Binary ROP (ROP2) code that gets executed by the hardware for each foreground pixel
background_rop	A Binary ROP (ROP2) code that gets executed by the hardware for each background pixel

Function

n2d_error_t n2d_draw_state()

n2d_fill Function

Fill a (partial) buffer with a specified color

File

libnano2D.h

C

```
n2d_error_t n2d_fill(n2d_buffer_t * destination, n2d_rectangle_t * rectangle, n2d_color_t color,
n2d_blend_t blend);
```

Returns

Returns the status as defined by n2d_error_t

Description

Draws and fills a rectangle with a specific color onto destination buffer.

An optional blend mode can be specified that defines the blending of the color onto the destination.

Remarks

This function will wait until the hardware is complete, i.e. it is synchronous

Parameters

Parameters	Description
destination	Pointer to a n2d_buffer_t structure that describes the buffer to be filled
rectangle	Pointer to a rectangle that specifies the area to be filled. If rectangle is NULL, the entire buffer will be filled with the specified color
color	The color value to use for filling the buffer
blend	The blending mode to be applied to each pixel. If no blending is required, set this value to N2D_BLEND_NONE (0)

Function

n2d_error_t n2d_fill()

n2d_init_hardware Function

Initializes the n2d driver and peripheral hardware

File

libnano2D.h

C

```
n2d_error_t n2d_init_hardware(n2d_module_parameters_t * params);
```

Returns

Returns the status as defined by n2d_error_t

Description

The initializes the memory region, sets base address, establishes the irq and connects the hardware to application.

Remarks

For PIC32MZ DA, registerMemBase2D is 0xBF8EB000 and baseAddress is 0

Parameters

Parameters	Description
params	Initialization parameters. See n2d_module_parameters_t

Function

n2d_error_t n2d_init_hardware()

n2d_line Function

Draw a line

File

libnano2D.h

C

```
n2d_error_t n2d_line(n2d_buffer_t * destination, n2d_point_t start, n2d_point_t end, n2d_rectangle_t *
clip, n2d_color_t color, n2d_blend_t blend);
```

Returns

Returns the status as defined by n2d_error_t

Description

Draw a line with a specific color. The last pixel of the line will not be drawn.

An optional blend mode can be specified that defines the blending of the color onto the destination.

Remarks

This function will wait until the hardware is complete, i.e. it is synchronous

Parameters

Parameters	Description
destination	Pointer to a n2d_buffer_t structure that describes the buffer to be used to draw the line into.
start	The starting point of the line, given in destination coordinates.
end	The ending point of the line, given in destination coordinates. The last pixel will not be drawn.
clip	Optional pointer to a rectangle that specifies the clipping region of the destination. If clip is NULL, the clipping region will be the entire destination buffer.
color	The color value to use for drawing the line.
blend	The blending mode to be applied to each pixel on the line. If no blending is required, set this value to N2D_BLEND_NONE (0).

Function

n2d_error_t n2d_line()

n2d_open Function

Open Nano2d context

File

libnano2D.h

C

```
n2d_error_t n2d_open();
```

Returns

Returns the status as defined by n2d_error_t.

Description

The n2d_line, n2d_fill, n2d_blit, and n2d_draw_state functions require a Nano2D context to be opened. This function is the first interface that accesses the hardware. The hardware will be turned on and initialized.

Remarks

There is only one Nano2d context per application, so this function must be called once in your application.

Function

n2d_error_t n2d_open()

n2d_dither Function

Enable or disable dithering

File

libnano2D.h

C

```
n2d_error_t n2d_dither(n2d_bool_t enable);
```

Returns

Returns the status as defined by n2d_error_t

Description

Sets the capability to scatter or approximate colors when using less than 32bpp or 16bpp color depth. Dither attempts to improve the overall appearance of low resolution images. Dithering is on when enable is true, otherwise, dithering is off.

Remarks

This function will wait until the hardware is complete, i.e. it is synchronous.

Parameters

Parameters	Description
enable	defines whether dither is set on or off.

Function

n2d_error_t n2d_dither()

b) Data Types and Constants

n2d_blend_t Enumeration

List of blending modes

File

libnano2D.h

C

```
typedef enum n2d_blend {
  N2D_BLEND_NONE,
  N2D_BLEND_SRC_OVER,
  N2D_BLEND_DST_OVER,
  N2D_BLEND_SRC_IN,
  N2D_BLEND_DST_IN,
  N2D_BLEND_ADDITIVE,
  N2D_BLEND_SUBTRACT
} n2d_blend_t;
```

Description

Structure: n2d_blend

N2D_BLEND_NONE - S, i.e. no blending N2D_BLEND_SRC_OVER - S + (1 - Sa) * D N2D_BLEND_DST_OVER - (1 - Da) * S + D N2D_BLEND_SRC_IN - Da * S N2D_BLEND_DST_IN - Sa * D N2D_BLEND_ADDITIVE - S + D N2D_BLEND_SUBTRACT - D * (1 - S)

Remarks

Some of the Nano2D API functions calls support blending. S and D represent source and destination color channels and Sa and Da represent the source and destination alpha channels

n2d_buffer_format_t Enumeration

List of blending modes

File

libnano2D.h

C

```
typedef enum n2d_buffer_format {
  N2D_RGBA8888,
  N2D_BGRA8888,
  N2D_RGB565,
  N2D_BGR565,
  N2D_RGBA4444,
  N2D_BGRA4444,
  N2D_A8
} n2d_buffer_format_t;
```

Members

Members	Description
N2D_RGBA4444	currently not available in MPLAB Harmony HAL
N2D_BGRA4444	currently not available in MPLAB Harmony HAL

Description

Structure: n2d_buffer_format

N2D_RGBA8888 - 32-bit RGBA format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16, and the alpha channel is in bits 31:24 N2D_BGRA8888 - 32-bit RGBA format with 8 bits per color channel. Red is in bits 23:16, green in bits 15:8, blue in bits 7:0, and the alpha channel is in bits 31:24 N2D_RGB565 - 16-bit RGB format with 5 and 6 bits per color channel. Red is in bits 4:0, green in bits 10:5, and the blue color channel is in bits 15:11 N2D_BGR565 - 16-bit RGB format with 5 and 6 bits per color channel. Red is in bits 15:11, green in bits 10:5, and the blue color channel is in bits 4:0 N2D_RGBA4444 - 16-bit RGBA format with 4 bits per color channel. Red is in bits 3:0, green in bits 7:4, blue in bits 11:8 and the alpha channel is in bits 15:12. Note: currently not available in HAL N2D_BGRA4444 - 16-bit RGBA format with 4 bits per color channel. Red is in bits 11:8, green in bits 7:4, blue in bits 3:0 and the alpha channel is in bits 15:12. Note: currently not available in HAL N2D_A8 - 8-bit alpha format. There are no RGB values.

Remarks

The pixel type for a n2d_buffer_t structure

n2d_buffer_t Structure

A wrapper structure for any image or render target

File

libnano2D.h

C

```
typedef struct n2d_buffer {
  n2d_int32_t width;
  n2d_int32_t height;
  n2d_int32_t stride;
  n2d_buffer_format_t format;
  n2d_orientation_t orientation;
  void * handle;
  void * memory;
  n2d_uint32_t gpu;
} n2d_buffer_t;
```

Description

Structure: n2d_buffer

width - Width of the buffer in pixels height - Height of the buffer in pixels stride - Stride of the buffer in bytes format - Pixel format of the buffer orientation - Buffer's orientation memory - Logical pointer to the buffer's memory for the CPU gpu - Physical address of the buffer's memory the hardware can access

Remarks

Each piece of memory, whether it is an image used as a source or a buffer used as a destination, requires a structure to define it. This structure contains all the information the Nano2D API requires to access the buffer's memory by the hardware

n2d_color_t Type

Identifies a specific pixel color

File

libnano2D.h

C

```
typedef n2d_int32_t n2d_color_t;
```

Description

Color

Color container

n2d_error_t Enumeration

Error codes that the Nano2D functions can return

File

libnano2D.h

C

```
typedef enum n2d_error {
  N2D_SUCCESS = 0,
  N2D_INVALID_ARGUMENT,
  N2D_OUT_OF_MEMORY,
  N2D_NO_CONTEXT,
  N2D_TIMEOUT,
  N2D_OUT_OF_RESOURCES,
  N2D_GENERIC_IO,
  N2D_NOT_SUPPORTED
} n2d_error_t;
```

Description

Structure: n2d_error

N2D_SUCCESS - Success N2D_INVALID_ARGUMENT - An invalid argument was specified N2D_OUT_OF_MEMORY - Out of memory N2D_NO_CONTEXT - No open context is present N2D_TIMEOUT - A timeout has accored during a wait N2D_OUT_OF_RESOURCES - Out of system resources N2D_GENERIC_IO - Cannot communicate with the kernel driver N2D_NOT_SUPPORTED - The request is not supported

Remarks

All API functions return a status code. On success, N2D_SUCCESS will be returned when a function is successful. This value is set to zero, so if any function returns a non-zero value, an error has occured

n2d_module_parameters_t Structure

GPU peripheral Initialization parameters

File

libnano2D.h

C

```
typedef struct n2d_module_parameters {
  n2d_int32_t irqLine2D;
  n2d_uint32_t registerMemBase2D;
  n2d_uint32_t registerMemSize2D;
  n2d_uint32_t contiguousSize;
  n2d_uint32_t contiguousBase;
  n2d_uint32_t baseAddress;
} n2d_module_parameters_t;
```

Description

Structure: n2d_module_parameters

irqLine2D - command completion interrupt pin registerMemBase2D - base address of gpu (physical address) registerMemsize2D - size of gpu address space contiguousSize - size of memory pool contiguousBase - start address of memory (virtual address) baseAddress - base address display buffer

Remarks

None

n2d_orientation_t Enumeration

List of blending modes

File

libnano2D.h

C

```
typedef enum n2d_orientation {
  N2D_0,
  N2D_90,
  N2D_180,
  N2D_270
} n2d_orientation_t;
```

Description

Structure: n2d_orientation

N2D_0 - Buffer is 0 degrees rotated. N2D_90 - Buffer is 90 degrees rotated. N2D_180 - Buffer is 180 degrees rotated. N2D_270 - Buffer is 270 degrees rotated.

Remarks

Orientation is orthogonal. Rotation which is not parallel to the x or y axis is not supported.

n2d_point_t Structure

A position on a pixel

File

libnano2D.h

C

```
typedef struct n2d_point {
  n2d_int32_t x;
  n2d_int32_t y;
} n2d_point_t;
```

Description

Structure: n2d_point

Defines a position on the screen

x - horizontal coordinate of the point y - vertical coordinate of the point

n2d_rectangle_t Structure

A rectangle

File

libnano2D.h

C

```
typedef struct n2d_rectangle {
  n2d_int32_t x;
  n2d_int32_t y;
  n2d_int32_t width;
  n2d_int32_t height;
} n2d_rectangle_t;
```

Description

Structure: n2d_rectangle

Defines a rectangular shape area of the screen

x - Left coordinate of the rectangle y - Top coordinate of the rectangle width - Width of the rectangle height - Height of the rectangle

n2d_transparency_t Enumeration

Transparency modes

File

libnano2D.h

C

```
typedef enum n2d_transparency {
  N2D_TRANSPARENCY_NONE,
```

```
    N2D_TRANSPARENCY_SOURCE,
    N2D_TRANSPARENCY_DESTINATION
} n2d_transparency_t;
```

Description

Structure: n2d_transparency

N2D_TRANSPARENCY_NONE - No transparency N2D_TRANSPARENCY_SOURCE - The source defines the transparency N2D_TRANSPARENCY_DESTINATION - The destination defines the transparency

Remarks

The Nano2D hardware can be programmed to use transparency, extracted from either the source or the destination

n2d_bool_t Type

File

libnano2D_types.h

C

```
typedef int n2d_bool_t;
```

Description

This is type n2d_bool_t.

n2d_float_t Type

File

libnano2D_types.h

C

```
typedef float n2d_float_t;
```

Description

This is type n2d_float_t.

n2d_int16_t Type

File

libnano2D_types.h

C

```
typedef short n2d_int16_t;
```

Description

This is type n2d_int16_t.

n2d_int32_t Type

File

libnano2D_types.h

C

```
typedef int n2d_int32_t;
```

Description

This is type n2d_int32_t.

n2d_size_t Type

File

libnano2D_types.h

C

```
typedef unsigned int n2d_size_t;
```

Description

This is type n2d_size_t.

n2d_uint16_t Type

File

libnano2D_types.h

C

```
typedef unsigned short n2d_uint16_t;
```

Description

This is type n2d_uint16_t.

n2d_uint32_t Type

File

libnano2D_types.h

C

```
typedef unsigned int n2d_uint32_t;
```

Description

This is type n2d_uint32_t.

n2d_uint64_t Type

File

libnano2D_types.h

C

```
typedef unsigned long long n2d_uint64_t;
```

Description

This is type n2d_uint64_t.

n2d_uint8_t Type

File

libnano2D_types.h

C

```
typedef unsigned char n2d_uint8_t;
```

Description

This is type n2d_uint8_t.

__gcmALIGN Macro

File

libnano2D_types.h

C

```
#define __gcmALIGN(data, reg_field) \
    (((n2d_uint32_t) (data)) << __gcmSTART(reg_field))
```

Description

This is macro __gcmALIGN.

__gcmEND Macro

File

libnano2D_types.h

C

```
#define __gcmEND(reg_field) \
    (1 ? reg_field)
```

Description

This is macro __gcmEND.

__gcmGETSIZE Macro

File

libnano2D_types.h

C

```
#define __gcmGETSIZE(reg_field) \
    (__gcmEND(reg_field) - __gcmSTART(reg_field) + 1)
```

Description

This is macro __gcmGETSIZE.

__gcmMASK Macro

File

libnano2D_types.h

C

```
#define __gcmMASK(reg_field) \
    ((n2d_uint32_t) ((__gcmGETSIZE(reg_field) == 32) \
        ?  ~0 \
        : (~(~0 << __gcmGETSIZE(reg_field)))))
```

Description

This is macro __gcmMASK.

__gcmSTART Macro

File

libnano2D_types.h

C

```
#define __gcmSTART(reg_field) \
    (0 ? reg_field)
```

Description

This is macro __gcmSTART.

_nano2D_types_h__ Macro

File

libnano2D_types.h

C

```
#define _nano2D_types_h__
```

Description

This is macro _nano2D_types_h__.

gcmALIGN Macro

File

libnano2D_types.h

C

```
#define gcmALIGN(n, align) \
( \
    ((n) + ((align) - 1)) & ~((align) - 1) \
)
```

Description

This is macro gcmALIGN.

gcmCOUNTOF Macro

File

libnano2D_types.h

C

```
#define gcmCOUNTOF(array) \
        (sizeof(array) / sizeof(array[0]))
```

Description

This is macro gcmCOUNTOF.

gcmGETFIELD Macro

File

libnano2D_types.h

C

```
#define gcmGETFIELD(data, reg, field) \
( \
    ((((n2d_uint32_t) (data)) >> __gcmSTART(reg##_##field)) \
        & __gcmMASK(reg##_##field)) \
)
```

Description

This is macro gcmGETFIELD.

gcmINT2PTR Macro

File

libnano2D_types.h

C

```
#define gcmINT2PTR(i) \
( \
    (void *)(n2d_uint32_t)(i) \
)
```

Description

This is macro gcmINT2PTR.

gcmMAX Macro

File

libnano2D_types.h

C

```
#define gcmMAX(x, y) \
( \
    ((x) >= (y)) \
        ? (x) \
        : (y) \
)
```

Description

This is macro gcmMAX.

gcmMIN Macro

File

libnano2D_types.h

C

```
#define gcmMIN(x, y) \
( \
    ((x) <= (y)) \
        ? (x) \
```

```
        : (y) \
)
```

Description

This is macro gcmMIN.

gcmPTR2INT Macro

File

libnano2D_types.h

C

```
#define gcmPTR2INT(p) \
( \
    (n2d_uint32_t)(p) \
)
```

Description

This is macro gcmPTR2INT.

gcmSETFIELD Macro

File

libnano2D_types.h

C

```
#define gcmSETFIELD(data, reg, field, value) \
( \
    (((n2d_uint32_t) (data)) \
        & ~__gcmALIGN(__gcmMASK(reg##_##field), reg##_##field)) \
        |  __gcmALIGN((n2d_uint32_t) (value) \
            & __gcmMASK(reg##_##field), reg##_##field) \
)
```

Description

This is macro gcmSETFIELD.

gcmSETFIELDVALUE Macro

File

libnano2D_types.h

C

```
#define gcmSETFIELDVALUE(data, reg, field, value) \
( \
    (((n2d_uint32_t) (data)) \
        & ~__gcmALIGN(__gcmMASK(reg##_##field), reg##_##field)) \
        |  __gcmALIGN(reg##_##field##_##value \
            & __gcmMASK(reg##_##field), reg##_##field) \
)
```

Description

This is macro gcmSETFIELDVALUE.

gcmSETMASKEDFIELD Macro

File

libnano2D_types.h

C

```
#define gcmSETMASKEDFIELD(reg, field, value) \
( \
    gcmSETFIELD     (~0, reg,          field, value) & \
    gcmSETFIELDVALUE(~0, reg, MASK_ ## field, ENABLED) \
)
```

Description

This is macro gcmSETMASKEDFIELD.

gcmSETMASKEDFIELDVALUE Macro

File

libnano2D_types.h

C

```
#define gcmSETMASKEDFIELDVALUE(reg, field, value) \
( \
    gcmSETFIELDVALUE(~0, reg,          field, value) & \
    gcmSETFIELDVALUE(~0, reg, MASK_ ## field, ENABLED) \
)
```

Description

This is macro gcmSETMASKEDFIELDVALUE.

gcmVERIFYFIELDVALUE Macro

File

libnano2D_types.h

C

```
#define gcmVERIFYFIELDVALUE(data, reg, field, value) \
( \
    (((n2d_uint32_t) (data)) >> __gcmSTART(reg##_##field) & \
                            __gcmMASK(reg##_##field)) \
        == \
    (reg##_##field##_##value & __gcmMASK(reg##_##field)) \
)
```

Description

This is macro gcmVERIFYFIELDVALUE.

IN Macro

File

libnano2D_types.h

C

```
#define IN
```

Description

This is macro IN.

N2D_FALSE Macro

File

libnano2D_types.h

C

```
#define N2D_FALSE 0
```

Description

This is macro N2D_FALSE.

N2D_INFINITE Macro

File

libnano2D_types.h

C

```
#define N2D_INFINITE ((n2d_uint32_t) ~0U)
```

Description

This is macro N2D_INFINITE.

N2D_IS_ERROR Macro

File

libnano2D_types.h

C

```
#define N2D_IS_ERROR(error) (error != N2D_SUCCESS)
```

Description

This is macro N2D_IS_ERROR.

N2D_IS_SUCCESS Macro

File

libnano2D_types.h

C

```
#define N2D_IS_SUCCESS(error) (error == N2D_SUCCESS)
```

Description

This is macro N2D_IS_SUCCESS.

N2D_NULL Macro

File

libnano2D_types.h

C

```
#define N2D_NULL ((void *) 0)
```

Description

This is macro N2D_NULL.

N2D_ON_ERROR Macro

File

libnano2D_types.h

C

```
#define N2D_ON_ERROR(func) \
    do \
    { \
        error = func; \
        if (N2D_IS_ERROR(error)) \
        { \
            goto on_error; \
        } \
    } \
    while (0)
```

Description

This is macro N2D_ON_ERROR.

N2D_TRUE Macro

File

libnano2D_types.h

C

```
#define N2D_TRUE 1
```

Description

This is macro N2D_TRUE.

OUT Macro

File

libnano2D_types.h

C

```
#define OUT
```

Description

This is macro OUT.

Files

This section describes the Application Programming Interface (API) functions of the Nano2D module.

Files

Name	Description
libnano2D.h	Main header file for MPLAB Harmony Graphics Driver libnano2D GPU functions
libnano2D_types.h	Defines libnano2D data types and constants

Description

Source File Name	Description
`libnano2D.h`	Header file that includes all of the nano2d APIs
`libnano2D_types.h`	Header file that includes all the nano2d data types
`libnano2D.a`	Static library file which must be linked to the application. This library implements all Nano2D feature APIs

libnano2D.h

Main header file for MPLAB Harmony Graphics Driver libnano2D GPU functions

Enumerations

	Name	Description
	n2d_blend	List of blending modes
	n2d_buffer_format	List of blending modes
	n2d_error	Error codes that the Nano2D functions can return
	n2d_orientation	List of blending modes
	n2d_transparency	Transparency modes
	n2d_blend_t	List of blending modes
	n2d_buffer_format_t	List of blending modes
	n2d_error_t	Error codes that the Nano2D functions can return
	n2d_orientation_t	List of blending modes
	n2d_transparency_t	Transparency modes

Functions

	Name	Description
	n2d_blit	Copy a source buffer to the the destination buffer
	n2d_dither	Enable or disable dithering
	n2d_draw_state	Set the drawing state for any following Nano2D API draw call
	n2d_fill	Fill a (partial) buffer with a specified color
	n2d_init_hardware	Initializes the n2d driver and peripheral hardware
	n2d_line	Draw a line
	n2d_open	Open Nano2d context

Structures

	Name	Description
	n2d_buffer	A wrapper structure for any image or render target
	n2d_module_parameters	GPU peripheral Initialization parameters

	n2d_point	A position on a pixel
	n2d_rectangle	A rectangle
	n2d_buffer_t	A wrapper structure for any image or render target
	n2d_module_parameters_t	GPU peripheral Initialization parameters
	n2d_point_t	A position on a pixel
	n2d_rectangle_t	A rectangle

Types

	Name	Description
	n2d_color_t	Identifies a specific pixel color

Description

Module for Microchip Graphics Library - Graphics Driver Layer

The API functions to be used for the Nano2D graphics accelerator.

File Name

libnano2D.h

Company

Microchip Technology Inc.

libnano2D_types.h

Defines libnano2D data types and constants

Macros

	Name	Description
	__gcmALIGN	This is macro __gcmALIGN.
	__gcmEND	This is macro __gcmEND.
	__gcmGETSIZE	This is macro __gcmGETSIZE.
	__gcmMASK	This is macro __gcmMASK.
	__gcmSTART	This is macro __gcmSTART.
	_nano2D_types_h__	This is macro _nano2D_types_h__.
	gcmALIGN	This is macro gcmALIGN.
	gcmCOUNTOF	This is macro gcmCOUNTOF.
	gcmGETFIELD	This is macro gcmGETFIELD.
	gcmINT2PTR	This is macro gcmINT2PTR.
	gcmMAX	This is macro gcmMAX.
	gcmMIN	This is macro gcmMIN.
	gcmPTR2INT	This is macro gcmPTR2INT.
	gcmSETFIELD	This is macro gcmSETFIELD.
	gcmSETFIELDVALUE	This is macro gcmSETFIELDVALUE.
	gcmSETMASKEDFIELD	This is macro gcmSETMASKEDFIELD.
	gcmSETMASKEDFIELDVALUE	This is macro gcmSETMASKEDFIELDVALUE.
	gcmVERIFYFIELDVALUE	This is macro gcmVERIFYFIELDVALUE.
	IN	This is macro IN.
	N2D_FALSE	This is macro N2D_FALSE.
	N2D_INFINITE	This is macro N2D_INFINITE.
	N2D_IS_ERROR	This is macro N2D_IS_ERROR.
	N2D_IS_SUCCESS	This is macro N2D_IS_SUCCESS.
	N2D_NULL	This is macro N2D_NULL.
	N2D_ON_ERROR	This is macro N2D_ON_ERROR.
	N2D_TRUE	This is macro N2D_TRUE.
	OUT	This is macro OUT.

Types

	Name	Description
	n2d_bool_t	This is type n2d_bool_t.

	n2d_float_t	This is type n2d_float_t.
	n2d_int16_t	This is type n2d_int16_t.
	n2d_int32_t	This is type n2d_int32_t.
	n2d_size_t	This is type n2d_size_t.
	n2d_uint16_t	This is type n2d_uint16_t.
	n2d_uint32_t	This is type n2d_uint32_t.
	n2d_uint64_t	This is type n2d_uint64_t.
	n2d_uint8_t	This is type n2d_uint8_t.

Description

Module for Microchip Graphics Library - Graphics Driver Layer

This is a definition file for libnano2d types and constants used in the library and applications.

File Name

libnano2D_types.h

Company

Microchip Technology Inc.

Graphics Utilities Library

This section provides information about the Graphics Utilities Library within the MPLAB Harmony framework for graphics applications.

Description

This library is primarily responsible for managing and decoding assets such as images, fonts, and strings. It provides the means for interacting with asset data, complex data decoding, data decompression, and string asset lookup. The library also abstractly handles the accessing of external memory sources during asset decoding.

Definitions

- Anti-aliasing – Uses transparency to achieve a less jagged look when rasterizing pixel information.
- ASCII – The standard 8-bit-per-character text representation.
- Asset – A generic term for any resource that consists of blocks of binary data. Can be images, raw files, fonts, strings, and so on.
- Binary Asset – A chunk of raw data.
- Codepoint – The numerical ID of a glyph, typically four bytes in size
- External Asset – An asset that is stored on an external storage peripheral not directly accessible from the CPU.
- Font – A collection of images that represent linguistic characters. Each font has a distinct look and feel.
- Glyph – A linguistic character.
- Image Asset – A collection of pixel data that, when rendered, forms a visual image.
- Index Map – An image that is stored as a series of lookup table indices, rather than raw pixel data.
- Palette Asset – A form of image compression. Palettes are lookup tables of color information.
- Run-length Encoding (RLE) – A simple form of data compression that indexes long strings of duplicate bytes into a single value with an associated length value. Data blocks with long runs of duplicate characters are good candidates for RLE compression. Poor candidates will see the data size increase rather than decrease.
- String – A series of characters often used to form linguistic sentences.
- UTF8 – The encoding format for Unicode characters that favors space over decoding speed.
- UTF16 – The encoding format for Unicode characters that favors decoding speed over space.

Graphics Utilities Library Objective

The library serves four main purposes:

- Provides a common definition for asset description.
- Provides APIs for asset indexing, decoding, and rendering.
- Provides an abstract API for asset decoders.
- Provides state machines for accessing assets located in external memory locations.

Asset Common Definition

All assets share a common header “GFXU_AssetHeader”. This header is defined as follows:

```
typedef struct GFXU_AssetHeader_t
{
 uint32_t type;
 uint32_t dataLocation;
```

```
 void* dataAddress;
 uint32_t dataSize;
} GFXU_AssetHeader;
```

- type – the type of an asset
- dataLocation – The location ID for the asset. A location of “0” always indicates internal flash memory.
- dataAddress – The address of the asset. Depending on memory location, this address may not be accessible from the CPU.
- dataSize – The size of the asset in bytes.

Image Decoding and Rendering

Image Asset Descriptor

The image asset descriptor is defined as follows:

```
typedef struct GFXU_ImageAsset_t
{
 GFXU_AssetHeader header;
 GFXU_ImageFormat format;
 uint32_t width;
 uint32_t height;
 GFX_ColorMode colorMode;
 GFXU_ImageCompressionType compType;
 GFX_Bool useMask;
 GFX_Color mask;
 GFXU_PaletteAsset* palette;
} GFXU_ImageAsset;
```

- header – The common asset header.
- format – The format of the image data.
- width – The width of the image.
- height – The height of the image.
- colorMode – The format of the image's pixel data.
- compType – If compressed, indicates the compression type.
- useMask – Indicates whether the image specifies a pixel transparency mask.
- mask – The value of the image transparency mask.
- palette – If the color mode is an index format, then this is the address of the lookup table to reference.

Image decoding with the GFX Utilities library is meant to be simple and straightforward. The library provides a single API that clients can use to render image assets.

```
GFX_Result GFXU_DrawImage(GFXU_ImageAsset* img,
 int32_t src_x,
 int32_t src_y,
 int32_t src_width,
 int32_t src_height,
 int32_t dest_x,
 int32_t dest_y,
 GFXU_MemoryIntf* read_cb,
 GFXU_ExternalAssetReader** reader);
```

This function accepts a pointer to an image header and rendering dimensions for rendering sub-sections of the image. The last two arguments refer to external memory access and will be described later. If the type of an image specifies an image decoder, then that decoder is automatically invoked by the library. If an image is stored as an index map, then its associated palette is referenced during decoding.

Image Palette Asset

Palette assets are color lookup tables that can be referenced when decoding indexed images. The index format can be 1bpp, 4bpp, or 8bpp large, resulting in a maximum of 1, 16, or 256 colors, respectively.

The palette descriptor is defined as follows:

```
typedef struct GFXU_PaletteAsset_t
{
 GFXU_AssetHeader header;
 uint32_t colorCount;
 GFX_ColorMode colorMode;
} GFXU_PaletteAsset;
```

- header – The common asset header.
- colorCount – The number of colors in this palette.
- colorMode – The color format of this palette.

Font Assets

Fonts are chunks of data that contain color information for drawing individual linguistic characters. Font glyph data can either be stored by using a 1bpp or 8bpp format. The larger format is for storing transparency information for use in drawing anti-aliased characters.

The font descriptor is as follows:

```
typedef struct GFXU_FontAsset_t
{
 GFXU_AssetHeader header;
 uint32_t height;
 uint32_t ascent;
 uint32_t descent;
 uint32_t baseline;
 GFXU_FontAssetBPP bpp;
 GFXU_FontGlyphIndexTable* indexTable;
} GFXU_FontAsset;
```

- header – The common asset header.
- height – The height of the font.
- ascent – The ascent of the font.
- descent – The descent of the font.
- baseline – The baseline of the font.
- bpp – The size of the per-pixel font data.
- indexTable – The pointer to the font index table.

Font Index Table

Each font provides an index table for quickly locating pixel data inside the font data chunk. The index table specifies the number of individual ranges or series of glyphs that it provides, followed by the actual range data.

A typical font range table entry is as follows:

```
typedef struct GFXU_FontGlyphRange_t
{
 uint32_t glyphCount;
 uint32_t startID;
 uint32_t endID;
 uint8_t* lookupTable;
} GFXU_FontGlyphRange;
```

- glyphCount – The number of glyphs in this range.
- startID – The starting glyph codepoint.
- endID – The ending glyph codepoint.
- lookupTable – The pointer to the lookup table for this range.

Font Lookup Table

The font lookup table contains location and size data for referencing glyphs in the font data chunk. This table provides with values to allow geometric analysis of font glyphs without having to render any data.

The data in a font lookup table is defined as follows:

- Byte 1 – The size of the offset values in the lookup table. 1-4 bytes possible depending on the max size of the font data chunk.
- Byte 2 – The size of the width values in the lookup table. This depends on the font size. The maximum size is 0xFFFF.

Repeating 'glyphCount' number of times:

- Offset value – Read offset value of 1-4 bytes, depending on the header.
- Width value – Read width value of 1-2 bytes, depending on the header

Font Glyph Raster Data

Font raster data is stored in a single chunk of memory and is referenced through the previously mentioned lookup tables. When rendering a font, the decoder uses the code point to find the appropriate lookup table and then uses that table to get the offset of the glyph. The width value says how large the glyph is in pixels, which could be 1bpp or 8bpp, depending on the anti-alias setting. The decoder then reads "width" number of pixels starting at "offset". The pixel data offsets are always byte-aligned.

String Table

The graphics utilities library defines a special asset called the "String Table". This table is a predefined lookup table of strings, languages, and their associated fonts. This construct makes runtime localization possible for user interface libraries.

Graphics Utilities Interface

a) Functions

	Name	Description
	convertColorAndSetDraw	internal use only
	getDiscreteValueAtIndex	internal use only
	getOffsetFromIndexAndBPP	internal use only
	getRLEDataAtIndex	internal use only
	GFXU_CalculateCharStringWidth	Gets the width of a string buffer in pixels.
	GFXU_CalculatePartialCharStringWidth	Gets the width of a partial string buffer in pixels.

	GFXU_CalculatePartialStringWidth	Gets the partial width, starting from the left, of a string contained in the string table.
	GFXU_CalculateStringWidth	Gets the width of a string contained in the string table.
	GFXU_CompareString	Compares a string table entry and a GFXU_CHAR type string.
	GFXU_DecodeCodePoint	internal use only
	GFXU_DecodeUTF16	internal use only
	GFXU_DecodeUTF8	internal use only
	GFXU_DrawCharString	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawImage	Draws a portion of the given image at the specified coordinates.
	GFXU_DrawString	Draws a string at the given coordinates. Strings are drawn from the top down.
	GFXU_ExtractString	Extracts a string from the string table into a local character buffer. The local buffer must have already been allocated. Will attempt to write 'size - 1' number of characters into the buffer with the last being a zero terminator.
	GFXU_FontGetGlyphInfo	internal use only
	GFXU_FontGetLookupTableEntry	internal use only
	GFXU_GetCharAt	Gets a character of a string contained in the string table.
	GFXU_GetCharWidth	Gets the width of a character for a given font.
	GFXU_GetStringHeight	Gets the height of a string contained in the string table.
	GFXU_GetStringLength	Gets the length of a string contained in a string table.
	GFXU_GetStringRect	Gets the bounding rectangle for a string contained in the string table.
	GFXU_GetStringSizeInBytes	Gets the size of a string contained in a string table, in bytes.
	GFXU_PaletteGetColor	Gets a color from a palette asset given an index value.
	GFXU_StringFontIndexLookup	internal use only
	GFXU_StringIndexLookup	internal use only
	GFXU_StringLookup	internal use only
	GFXU_DecodeAndDrawString	internal use only
	GFXU_DrawCharStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawGlyph	internal use only
	GFXU_DrawGlyphRow	internal use only
	GFXU_DrawStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawUnknownGlyph	internal use only
	GFXU_GetGlyphRowDataSize	internal use only
	GFXU_PreprocessImage	Preprocesses an image to a specified memory address.
	GFXU_DecodeAndDrawSubString	internal use only
	GFXU_DrawCharSubStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawSubStringClipped	Draws a sub-string from a string asset in a clipped rectangle at the given coordinates. Strings are drawn from the top down.
	GFXU_GetCharStringLineRect	Gets the bounding rectangle for a line in a character string.
	GFXU_GetStringLineRect	Gets the bounding rectangle for a line in a string asset.

b) Data Types and Constants

	Name	Description
	GFXU_AssetHeader_t	Defines a common header for all assets supported by the generic decoder interface.
	GFXU_AssetType_t	Enumerates known asset types.
	GFXU_BinaryAsset_t	A binary asset type. Generic data that can be of any type
	GFXU_ExternalAssetReader_t	Defines the base description of an external asset reader. Specific decoder implementations will build on this foundation for each decoder type.
	GFXU_ExternalAssetReaderStatus_t	Enumerates external reader state machine states. Invalid - state machine hasn't been initialized Read - state machine is ready to begin processing but hasn't been run yet Waiting - state machine is waiting for a memory operation to complete Drawing - state machine is waiting for a drawing operation to complete Finished - state machine has finished drawing the entire asset Aborted - state machine encountered an error and has aborted
	GFXU_FontAsset_t	Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data. header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable -... more

	GFXU_FontAssetBPP	Indicates the bits per pixel mode of a font
	GFXU_FontGlyphIndexTable_t	Describes a font glyph index table. Essentially a series of glyph look up tables where each non-consecutive range of glyphs occupies one range entry count - number of ranges in the index table ranges - the glyph range array
	GFXU_FontGlyphRange_t	Describes a range of glyphs for a font. All IDs are in raw code points and are not encoded in any way glyphCount - number of glyphs in the range startID - the starting glyph id endID - the ending glyph id lookupTable - the corresponding look up table to find the glyph raster data
	GFXU_ImageAsset_t	Describes an image asset. header - standard asset header format - the format of the image. this directly affects which decoder is invoked when rendering the image width - the width of the image in pixels height - the height of the image in pixels colorMode - the color mode of the image compType - the compression mode of the image useMask - indicates of the mask field is used mask - may contain a masking color for the image. blit operations may reference this value and reject image pixels that match this value palette - will contain a valid... more
	GFXU_ImageCompressionType_t	Indicates the image compression type, only applies to RAW types
	GFXU_ImageFormat_t	Indicates the image encoding format
	GFXU_MemoryIntf_t	Defines a memory interface for all memory operations. Essentially wraps a GFX_MemoryIntf with the notable addition lf a GFXU_MemoryReadRequest_FnPtr. The 'read' function pointer will be invoked when a decoder attempts to access an asset that has a location that is greater than zero. The application must then look at the asset's location id and determine which peripheral to communicate with.
	GFXU_PaletteAsset_t	Describes a palette asset. A palette is a lookup table for unique colors. Given in an index, a color can be retrieved. header - standard asset header colorCount - the number of colors contained in the palette colorMode - the color mode of the image
	GFXU_StringEncodingMode_t	Indicates the string encoding mode type. Any characters above 255 must use UTF8 or UTF16
	GFXU_StringTableAsset_t	Describes a string table asset. There is typically only ever one of these defined at any one time. header - standard asset header languageCount - the number of languages in the string table stringCount - the number of strings in the string table stringIndexTable - the pointer to the string index table. the string index table is a table that contains all of the unique strings defined in the string table. fontTable - the font table contains an array of pointers to all defined font assets that the string table references fontIndexTable - the font index table is a table... more
	GFXU_AssetHeader	Defines a common header for all assets supported by the generic decoder interface.
	GFXU_AssetType	Enumerates known asset types.
	GFXU_BinaryAsset	A binary asset type. Generic data that can be of any type
	GFXU_CHAR	strings are defined as having 32bit characters due to the need to support international code points and unicode strings encoded using these types will not be compatible with standard library string functions like strlen or strcat
	GFXU_ExternalAssetReader	Defines the base description of an external asset reader. Specific decoder implementations will build on this foundation for each decoder type.
	GFXU_ExternalAssetReaderRun_FnPtr	This function pointer represents a function that maintains the state of an external reader state machine. The argument is the state machine to process.
	GFXU_ExternalAssetReaderStatus	Enumerates external reader state machine states. Invalid - state machine hasn't been initialized Read - state machine is ready to begin processing but hasn't been run yet Waiting - state machine is waiting for a memory operation to complete Drawing - state machine is waiting for a drawing operation to complete Finished - state machine has finished drawing the entire asset Aborted - state machine encountered an error and has aborted
	GFXU_FontAsset	Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data. header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable -... more
	GFXU_FontGlyphIndexTable	Describes a font glyph index table. Essentially a series of glyph look up tables where each non-consecutive range of glyphs occupies one range entry count - number of ranges in the index table ranges - the glyph range array

	GFXU_FontGlyphRange	Describes a range of glyphs for a font. All IDs are in raw code points and are not encoded in any way glyphCount - number of glyphs in the range startID - the starting glyph id endID - the ending glyph id lookupTable - the corresponding look up table to find the glyph raster data
	GFXU_ImageAsset	Describes an image asset. header - standard asset header format - the format of the image. this directly affects which decoder is invoked when rendering the image width - the width of the image in pixels height - the height of the image in pixels colorMode - the color mode of the image compType - the compression mode of the image useMask - indicates of the mask field is used mask - may contain a masking color for the image. blit operations may reference this value and reject image pixels that match this value palette - will contain a valid... more
	GFXU_ImageCompressionType	Indicates the image compression type, only applies to RAW types
	GFXU_ImageFormat	Indicates the image encoding format
	GFXU_MemoryIntf	Defines a memory interface for all memory operations. Essentially wraps a GFX_MemoryIntf with the notable addition lf a GFXU_MemoryReadRequest_FnPtr. The 'read' function pointer will be invoked when a decoder attempts to access an asset that has a location that is greater than zero. The application must then look at the asset's location id and determine which peripheral to communicate with.
	GFXU_PaletteAsset	Describes a palette asset. A palette is a lookup table for unique colors. Given in an index, a color can be retrieved. header - standard asset header colorCount - the number of colors contained in the palette colorMode - the color mode of the image
	GFXU_StringEncodingMode	Indicates the string encoding mode type. Any characters above 255 must use UTF8 or UTF16
	GFXU_ASSET_LOCATION_INTERNAL	This is macro GFXU_ASSET_LOCATION_INTERNAL.
	GFXU_STRING_ARRAY_SIZE	defines meta data sizes for the string table, don't change!
	GFXU_STRING_ENTRY_SIZE	This is macro GFXU_STRING_ENTRY_SIZE.
	GFXU_STRING_MAX_CHAR_WIDTH	This is macro GFXU_STRING_MAX_CHAR_WIDTH.
	GFXU_MediaCloseRequest_FnPtr	A callback that indicates that a media decoder is finished with a given media location and that the application can close if it. The argument is the asset that was being read.
	GFXU_MediaOpenRequest_FnPtr	A callback that indicates that a media decoder wishes to read from an external media source and that the application should prepare that source. The argument is the asset that needs to be read. If the result is false then the decoder will abort.
	GFXU_MediaReadRequest_FnPtr	callback
	GFXU_MediaReadRequestCallback_FnPtr	A callback that signifies that a media read request has been fulfilled. Often signals a state machine to continue processing a decode operation. The argument is the reader that requested the memory.

Description

This section describes the interface for the Graphics Utilities Library.

a) Functions

convertColorAndSetDraw Function

File

gfxu_image_utils.h

C

```
GFX_Result convertColorAndSetDraw(uint32_t color, GFX_ColorMode mode);
```

Description

internal use only

getDiscreteValueAtIndex Function

File

gfxu_image_utils.h

C

```
uint32_t getDiscreteValueAtIndex(uint32_t index, uint32_t value, GFX_ColorMode mode);
```

Description

internal use only

getOffsetFromIndexAndBPP Function

File

gfxu_image_utils.h

C

```
uint32_t getOffsetFromIndexAndBPP(uint32_t index, GFX_BitsPerPixel bpp);
```

Description

internal use only

getRLEDataAtIndex Function

File

gfxu_image_utils.h

C

```
uint32_t getRLEDataAtIndex(uint8_t* data, uint32_t max, uint32_t idx, uint32_t* startBlock, uint32_t*
startOffset);
```

Description

internal use only

GFXU_CalculateCharStringWidth Function

Gets the width of a string buffer in pixels.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_CalculateCharStringWidth(GFXU_CHAR* str, GFXU_FontAsset* fnt);
```

Returns

uint32_t - the width of the string buffer

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer, terimated with a zero value
GFXU_FontAsset* fnt	the font asset to reference

Function

uint32_t GFXU_CalculateCharStringWidth(GFXU_CHAR* str,
GFXU_FontAsset* fnt)

GFXU_CalculatePartialCharStringWidth Function

Gets the width of a partial string buffer in pixels.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_CalculatePartialCharStringWidth(GFXU_CHAR* str, GFXU_FontAsset* fnt, uint32_t length);
```

Returns

uint32_t - the width of the partial string buffer

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer, terimated with a zero value
GFXU_FontAsset* fnt	the font asset to reference
uint32_t length	the number of characters to include

Function

uint32_t GFXU_CalculatePartialCharStringWidth(GFXU_CHAR* str,
GFXU_FontAsset* fnt,
uint32_t length)

GFXU_CalculatePartialStringWidth Function

Gets the partial width, starting from the left, of a string contained in the string table.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_CalculatePartialStringWidth(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, uint32_t length);
```

Returns

uint32_t - the width of the sub-string

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
uint32_t length	the number of characters to include in the sub-string

Function

uint32_t GFXU_CalculatePartialStringWidth(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
uint32_t length)

GFXU_CalculateStringWidth Function

Gets the width of a string contained in the string table.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_CalculateStringWidth(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang);
```

Returns

uint32_t - the width of the string

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up

Function

uint32_t GFXU_CalculateStringWidth(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang)

GFXU_CompareString Function

Compares a string table entry and a GFXU_CHAR type string.

File

gfxu_string.h

C

```
LIB_EXPORT int32_t GFXU_CompareString(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, GFXU_CHAR*
buffer);
```

Returns

int32_t - the compare result, should be identical to strcmp()

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
GFXU_CHAR* buffer	char buffer to compare

Function

int32_t GFXU_CompareString(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
GFXU_CHAR* buffer)

GFXU_DecodeCodePoint Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_DecodeCodePoint(uint32_t encoding, uint8_t* data, uint32_t max, uint32_t* codePoint,
uint32_t* offset);
```

Description

internal use only

GFXU_DecodeUTF16 Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_DecodeUTF16(uint8_t* val, uint32_t max, uint32_t* codePoint, uint32_t* size);
```

Description

internal use only

GFXU_DecodeUTF8 Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_DecodeUTF8(uint8_t* val, uint32_t max, uint32_t* codePoint, uint32_t* size);
```

Description

internal use only

GFXU_DrawCharString Function

Draws a clipped string at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawCharString(GFXU_CHAR* str, GFXU_FontAsset* fnt, int32_t x, int32_t y,
GFXU_MemoryIntf* memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer to draw
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawCharString(GFXU_CHAR* str,
GFXU_FontAsset* fnt
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_DrawImage Function

Draws a portion of the given image at the specified coordinates.

File

gfxu_image.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawImage(GFXU_ImageAsset* img, int32_t src_x, int32_t src_y, int32_t src_width,
int32_t src_height, int32_t dest_x, int32_t dest_y, GFXU_MemoryIntf* read_cb, GFXU_ExternalAssetReader**
reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_ImageAsset* img	pointer to the image asset to draw
int32_t src_x	the x position of the source image to draw (0 if whole image)
int32_t src_y	the y position of the source image to draw (0 if whole image)
int32_t src_width	the width of the source rectangle to draw (source width if whole image) the height of the source rectangle to draw (source height if whole image)
int32_t dest_x	the x position to draw to
int32_t dest_y	the y position to draw to
GFXU_MemoryIntf* read_cb	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawImage(void);

GFXU_DrawString Function

Draws a string at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawString(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, int32_t x,
int32_t y, GFXU_MemoryIntf* memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawString(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_ExtractString Function

Extracts a string from the string table into a local character buffer. The local buffer must have already been allocated. Will attempt to write 'size - 1' number of characters into the buffer with the last being a zero terminator.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_ExtractString(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, GFXU_CHAR*
buffer, uint32_t size, uint32_t offset);
```

Returns

uint32_t - the number of characters written

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to reference
uint32_t lang	the id of the language to reference
GFXU_CHAR* buffer	the pointer to the buffer to write to
uint32_t size	the maximum size of the buffer to write to
uint32_t offset	the buffer write offset if any

Function

uint32_t GFXU_ExtractString(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
GFXU_CHAR* buffer,
uint32_t size,
uint32_t offset)

GFXU_FontGetGlyphInfo Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_FontGetGlyphInfo(GFXU_FontAsset* fnt, uint32_t glyph, uint32_t* offset, uint32_t* width);
```

Description

internal use only

GFXU_FontGetLookupTableEntry Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_FontGetLookupTableEntry(uint8_t* table, uint32_t index, uint32_t max, uint32_t* offset,
uint32_t* width);
```

Description

internal use only

GFXU_GetCharAt Function

Gets a character of a string contained in the string table.

File

gfxu_string.h

C

```
LIB_EXPORT GFXU_CHAR GFXU_GetCharAt(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, uint32_t idx);
```

Returns

GFXU_CHAR - the code point of the character

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
uint32_t idx	the index of the characer

Function

GFXU_CHAR GFXU_GetCharAt(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
uint32_t idx)

GFXU_GetCharWidth Function

Gets the width of a character for a given font.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetCharWidth(GFXU_CHAR chr, GFXU_FontAsset* fnt);
```

Returns

uint32_t - the width of the character or zero if the font doesn't contain that character

Parameters

Parameters	Description
GFXU_CHAR chr	the code point of the character
GFXU_FontAsset* fnt	the font to reference

Function

uint32_t GFXU_GetCharWidth(GFXU_CHAR chr, GFXU_FontAsset* fnt)

GFXU_GetStringHeight Function

Gets the height of a string contained in the string table.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetStringHeight(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang);
```

Returns

uint32_t - the height of the string

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up

Function

uint32_t GFXU_GetStringHeight(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang)

GFXU_GetStringLength Function

Gets the length of a string contained in a string table.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetStringLength(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang);
```

Returns

uint32_t - the length of the string entry

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up

Function

uint32_t GFXU_GetStringLength(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang)

GFXU_GetStringRect Function

Gets the bounding rectangle for a string contained in the string table.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_GetStringRect(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, GFX_Rect*
rect);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
GFX_Rect* rect	the resultant rectangle of the string

Function

GFX_Result GFXU_GetStringRect(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
GFX_Rect* rect)

GFXU_GetStringSizeInBytes Function

Gets the size of a string contained in a string table, in bytes.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetStringSizeInBytes(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang);
```

Returns

uint32_t - the size of the string entry in bytes

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up

Function

uint32_t GFXU_GetStringSizeInBytes(GFXU_StringTableAsset* tbl,

uint32_t id,

uint32_t lang)

GFXU_PaletteGetColor Function

Gets a color from a palette asset given an index value.

File

gfxu_palette.h

C

```
GFX_Color GFXU_PaletteGetColor(GFXU_PaletteAsset* pal, uint32_t idx, GFXU_MemoryIntf* read_cb,
GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Color - the color that was retrieved

Parameters

Parameters	Description
GFXU_PaletteAsset* pal	pointer to the palette to read
uint32_t idx	the index of the color to look up
GFXU_MemoryIntf* read_cb	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the palette asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_PaletteGetColor(void);

GFXU_StringFontIndexLookup Function

File

gfxu_string_utils.h

C

```
LIB_EXPORT GFXU_FontAsset* GFXU_StringFontIndexLookup(GFXU_StringTableAsset* table, uint32_t stringID,
uint32_t languageID);
```

Description

internal use only

GFXU_StringIndexLookup Function

File

gfxu_string_utils.h

C

```
uint16_t GFXU_StringIndexLookup(GFXU_StringTableAsset* table, uint32_t stringID, uint32_t languageID);
```

Description

internal use only

GFXU_StringLookup Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_StringLookup(GFXU_StringTableAsset* table, uint32_t stringIndex, uint8_t** stringAddress,
uint32_t* stringSize);
```

Description

internal use only

GFXU_DecodeAndDrawString Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_DecodeAndDrawString(uint8_t* string, uint32_t length, GFXU_StringEncodingMode mode,
GFXU_FontAsset* fnt, int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t x,
int32_t y);
```

Description

internal use only

GFXU_DrawCharStringClipped Function

Draws a clipped string at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawCharStringClipped(GFXU_CHAR* str, GFXU_FontAsset* fnt, int32_t clipX,
int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t x, int32_t y, GFXU_MemoryIntf*
memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer to draw
int32_t clipX	clipped x position
int32_t clipY	clipped y position
int32_t clipWidth	clipped rectangle width
int32_t clipHeight	clipped rectangle height
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawCharStringClipped(GFXU_CHAR* str,
GFXU_FontAsset* fnt
int32_t clipX,
int32_t clipY,
int32_t clipWidth,
int32_t clipHeight,
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_DrawGlyph Function

File

gfxu_string_utils.h

C

```
int32_t GFXU_DrawGlyph(GFXU_FontAsset* fnt, uint32_t glyph, int32_t clipX, int32_t clipY, int32_t
clipWidth, int32_t clipHeight, int32_t x, int32_t y);
```

Description

internal use only

GFXU_DrawGlyphRow Function

File

gfxu_string_utils.h

C

```
void GFXU_DrawGlyphRow(GFXU_FontAssetBPP bpp, uint8_t* data, int32_t width, int32_t x, int32_t y, int32_t
clipXStart, int32_t clipXEnd);
```

Description

internal use only

GFXU_DrawStringClipped Function

Draws a clipped string at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawStringClipped(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang,
int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t x, int32_t y, GFXU_MemoryIntf*
memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
int32_t clipX	clipped x position
int32_t clipY	clipped y position

int32_t clipWidth	clipped rectangle width
int32_t clipHeight	clipped rectangle height
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawStringClipped(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
int32_t clipX,
int32_t clipY,
int32_t clipWidth,
int32_t clipHeight,
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_DrawUnknownGlyph Function

File

gfxu_string_utils.h

C

```
int32_t GFXU_DrawUnknownGlyph(int32_t x, int32_t y, int32_t height, int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t clipHeight);
```

Description

internal use only

GFXU_GetGlyphRowDataSize Function

File

gfxu_string_utils.h

C

```
int32_t GFXU_GetGlyphRowDataSize(GFXU_FontAssetBPP bpp, int32_t width);
```

Description

internal use only

GFXU_PreprocessImage Function

Preprocesses an image to a specified memory address.

File

gfxu_image.h

C

```
LIB_EXPORT GFX_Result GFXU_PreprocessImage(GFXU_ImageAsset* img, uint32_t destAddress, GFX_ColorMode destMode, GFX_Bool padBuffer);
```

Returns

GFX_Result - the result of the operation

Description

This function preprocesses an image asset through the HAL pipeline and renders it to a given address, in a given color mode, and can pad the image buffer dimensions to be powers of 2 as required by some graphics accelerators.

This function is also useful for pre-staging images into run-time memory locations.

The caller is required to ensure that the destination address is capable of containing the result. The size can be calculated by using the method:

GFX_ColorInfo[destMode].size * img->width * img->height

This function only works with images that are located in a core accessible memory location like SRAM or DDR. If the image is located in an external source then GFXU_DrawImage should be called directly. The caller will then need to service the media streaing state machine. Once finished the image asset descriptor must be changed manually. This function can be used as a reference on how to accomplish this.

Parameters

Parameters	Description
GFXU_ImageAsset* img	pointer to the image asset to draw
uint32_t destAddress	the address to render the image to
GFX_ColorMode destMode	the desired output mode of the image
GFX_Bool padBuffer	indicates that the image buffer dimensions should be padded to equal powers of 2 (required by some GPUs)
GFXU_MemoryIntf* read_cb	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_PreprocessImage(GFXU_ImageAsset* img,
uint32_t destAddress,
GFX_ColorMode destMode,
GFX_Bool padBuffer);

GFXU_DecodeAndDrawSubString Function

File

gfxu_string_utils.h

C

```
GFX_Result GFXU_DecodeAndDrawSubString(uint8_t* string, uint32_t length, GFXU_StringEncodingMode mode,
GFXU_FontAsset* fnt, uint32_t start, uint32_t end, int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t
clipHeight, int32_t x, int32_t y);
```

Description

internal use only

GFXU_DrawCharSubStringClipped Function

Draws a clipped string at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawCharSubStringClipped(GFXU_CHAR* str, GFXU_FontAsset* fnt, uint32_t start,
uint32_t end, int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t x, int32_t y,
GFXU_MemoryIntf* memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer to draw

GFXU_FontAsset* fnt	font asset to use
uint32_t start	start offset of substring
uint32_t end	end offset of substring
int32_t clipX	clipped x position
int32_t clipY	clipped y position
int32_t clipWidth	clipped rectangle width
int32_t clipHeight	clipped rectangle height
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets
GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawCharSubStringClipped(GFXU_CHAR* str,
GFXU_FontAsset* fnt
uint32_t start,
uint32_t end,
int32_t clipX,
int32_t clipY,
int32_t clipWidth,
int32_t clipHeight,
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_DrawSubStringClipped Function

Draws a sub-string from a string asset in a clipped rectangle at the given coordinates. Strings are drawn from the top down.

File

gfxu_string.h

C

```
LIB_EXPORT GFX_Result GFXU_DrawSubStringClipped(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang,
uint32_t start, uint32_t end, int32_t clipX, int32_t clipY, int32_t clipWidth, int32_t clipHeight, int32_t
x, int32_t y, GFXU_MemoryIntf* memoryInterface, GFXU_ExternalAssetReader** reader);
```

Returns

GFX_Result

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
uint32_t start	offset of first character of substring
uint32_t end	offset of last character of substring
int32_t clipX	clipped x position
int32_t clipY	clipped y position
int32_t clipWidth	clipped rectangle width
int32_t clipHeight	clipped rectangle height
int32_t x	the x position to which to draw the string
int32_t y	the y position to which to draw the string
GFXU_MemoryIntf* memoryInterface	a pointer to a memory interface to use for memory operations, not needed for internal assets

GFXU_ExternalAssetReader** reader	will return as a valid pointer if the image asset is located in an external source. If this is the case then the caller is responsible for servicing the external asset reader state machine until completion. The caller is then responsible for freeing the reader's memory.

Function

GFX_Result GFXU_DrawSubStringClipped(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
uint32_t start,
uint32_t end,
int32_t clipX,
int32_t clipY,
int32_t clipWidth,
int32_t clipHeight,
int32_t x,
int32_t y,
GFXU_MemoryIntf* memoryInterface,
GFXU_ExternalAssetReader** reader)

GFXU_GetCharStringLineRect Function

Gets the bounding rectangle for a line in a character string.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetCharStringLineRect(GFXU_CHAR* str, GFXU_FontAsset* font, uint32_t offset,
GFX_Rect* rect);
```

Returns

The offset of end of line (including line feed or end of string)

Parameters

Parameters	Description
GFXU_CHAR* str	the string buffer, terimated with a zero value
GFXU_FontAsset* fnt	the font asset to reference
uint32_t offset	the start offset of the first character in the line
uint32_t length	the number of characters to include

Function

uint32_t GFXU_GetCharStringLineRect(GFXU_CHAR* str,
GFXU_FontAsset* font,
uint32_t offset,
GFX_Rect* rect)

GFXU_GetStringLineRect Function

Gets the bounding rectangle for a line in a string asset.

File

gfxu_string.h

C

```
LIB_EXPORT uint32_t GFXU_GetStringLineRect(GFXU_StringTableAsset* tbl, uint32_t id, uint32_t lang, uint32_t
offset, GFX_Rect* rect);
```

Returns

The offset of end of line (including line feed or end of string)

Parameters

Parameters	Description
GFXU_StringTableAsset* tbl	the string table to reference
uint32_t id	the id of the string to look up
uint32_t lang	the language entry to look up
uint32_t offset	the start offset of the first character in the line
GFX_Rect* rect	the resultant rectangle of the string

Function

uint32_t GFXU_GetStringLineRect(GFXU_StringTableAsset* tbl,
uint32_t id,
uint32_t lang,
uint32_t offset,
GFX_Rect* rect)

b) Data Types and Constants

GFXU_FontAsset_t Structure

Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data.

header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable - pointer to the corresponding glyph index table. this table is used to reference code points to glyph data.

File

gfxu_font.h

C

```
struct GFXU_FontAsset_t {
  GFXU_AssetHeader header;
  uint32_t height;
  uint32_t ascent;
  uint32_t descent;
  uint32_t baseline;
  GFXU_FontAssetBPP bpp;
  GFXU_FontGlyphIndexTable* indexTable;
};
```

Description

Structure: GFXU_FontAsset_t

GFXU_FontAssetBPP Enumeration

Indicates the bits per pixel mode of a font

File

gfxu_font.h

C

```
enum GFXU_FontAssetBPP {
  GFXU_FONT_BPP_1,
  GFXU_FONT_BPP_8
};
```

Members

Members	Description
GFXU_FONT_BPP_1	standard
GFXU_FONT_BPP_8	antialiased

Description

Enumeration: GFXU_FontAssetBPP

GFXU_AssetHeader Structure

Defines a common header for all assets supported by the generic decoder interface.

File

gfxu_global.h

C

```
typedef struct GFXU_AssetHeader_t {
  uint32_t type;
  uint32_t dataLocation;
  void* dataAddress;
  uint32_t dataSize;
} GFXU_AssetHeader;
```

Description

Structure: GFXU_AssetHeader_t

type - GFXU_AssetType - indicates the type of the asset dataLocation - indicates the location of the asset data. 0 is always internal flash. any other number must be understood by the application dataAddress - the address at which the data resides. may be a local pointer or a location in some external storage location not in the local memory map. dataSize - the size of the asset data in bytes

GFXU_AssetType Enumeration

Enumerates known asset types.

File

gfxu_global.h

C

```
typedef enum GFXU_AssetType_t {
  GFXU_ASSET_TYPE_IMAGE = 0,
  GFXU_ASSET_TYPE_PALETTE,
  GFXU_ASSET_TYPE_FONT,
  GFXU_ASSET_TYPE_BINARY,
  GFXU_ASSET_TYPE_STRINGTABLE
} GFXU_AssetType;
```

Description

enum: GFXU_AssetType_t

GFXU_BinaryAsset Structure

A binary asset type. Generic data that can be of any type

File

gfxu_binary.h

C

```
typedef struct GFXU_BinaryAsset_t {
  GFXU_AssetHeader header;
} GFXU_BinaryAsset;
```

Members

Members	Description
GFXU_AssetHeader header;	generic asset header

Description

Structure: GFXU_BinaryAsset_t

GFXU_CHAR Type

File

gfxu_string.h

C

```
typedef uint32_t GFXU_CHAR;
```

Description

strings are defined as having 32bit characters due to the need to support international code points and unicode strings encoded using these types will not be compatible with standard library string functions like strlen or strcat

GFXU_ExternalAssetReader Structure

Defines the base description of an external asset reader. Specific decoder implementations will build on this foundation for each decoder type.

File

gfxu_global.h

C

```
typedef struct GFXU_ExternalAssetReader_t {
  GFXU_ExternalAssetReaderStatus status;
  uint32_t state;
  int32_t result;
  GFXU_MemoryIntf* memIntf;
  GFXU_ExternalAssetReaderRun_FnPtr run;
  void* userData;
} GFXU_ExternalAssetReader;
```

Description

Structure: GFXU_MemoryIntf_t

status - the overall status of the decoder state machine state - mostly for decoder internal use result - can be used for an overall result of the state of an operation memIntf - the memory interface the decoder is using for memory operations run - the run pointer that must be called periodically to allow the state machine to process to completion.

GFXU_ExternalAssetReaderRun_FnPtr Type

This function pointer represents a function that maintains the state of an external reader state machine.

The argument is the state machine to process.

File

gfxu_global.h

C

```
typedef GFX_Result (* GFXU_ExternalAssetReaderRun_FnPtr)(GFXU_ExternalAssetReader*);
```

Description

typedef: GFXU_ExternalAssetReaderRun_FnPtr

GFXU_ExternalAssetReaderStatus Enumeration

Enumerates external reader state machine states.

Invalid - state machine hasn't been initialized Read - state machine is ready to begin processing but hasn't been run yet Waiting - state machine is waiting for a memory operation to complete Drawing - state machine is waiting for a drawing operation to complete Finished - state machine has finished drawing the entire asset Aborted - state machine encountered an error and has aborted

File

gfxu_global.h

C

```
typedef enum GFXU_ExternalAssetReaderStatus_t {
  GFXU_READER_STATUS_INVALID = 0,
```

```
  GFXU_READER_STATUS_READY,
  GFXU_READER_STATUS_WAITING,
  GFXU_READER_STATUS_DRAWING,
  GFXU_READER_STATUS_FINISHED,
  GFXU_READER_STATUS_ABORTED
} GFXU_ExternalAssetReaderStatus;
```

Description

enum: GFXU_ExternalAssetReaderStatus_t

GFXU_FontAsset Type

Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data.

header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable - pointer to the corresponding glyph index table. this table is used to reference code points to glyph data.

File

gfxu_string.h

C

```
typedef struct GFXU_FontAsset_t GFXU_FontAsset;
```

Description

Structure: GFXU_FontAsset_t

GFXU_FontGlyphIndexTable Structure

Describes a font glyph index table. Essentially a series of glyph look up tables where each non-consecutive range of glyphs occupies one range entry

count - number of ranges in the index table ranges - the glyph range array

File

gfxu_font.h

C

```
typedef struct GFXU_FontGlyphIndexTable_t {
  uint32_t count;
  GFXU_FontGlyphRange ranges[];
} GFXU_FontGlyphIndexTable;
```

Description

Structure: GFXU_FontGlyphIndexTable_t

GFXU_FontGlyphRange Structure

Describes a range of glyphs for a font. All IDs are in raw code points and are not encoded in any way

glyphCount - number of glyphs in the range startID - the starting glyph id endID - the ending glyph id lookupTable - the corresponding look up table to find the glyph raster data

File

gfxu_font.h

C

```
typedef struct GFXU_FontGlyphRange_t {
  uint32_t glyphCount;
  uint32_t startID;
  uint32_t endID;
  uint8_t* lookupTable;
} GFXU_FontGlyphRange;
```

Description

Structure: GFXU_FontGlyphRange_t

GFXU_ImageAsset Structure

Describes an image asset.

header - standard asset header format - the format of the image. this directly affects which decoder is invoked when rendering the image width - the width of the image in pixels height - the height of the image in pixels colorMode - the color mode of the image compType - the compression mode of the image useMask - indicates of the mask field is used mask - may contain a masking color for the image. blit operations may reference this value and reject image pixels that match this value palette - will contain a valid pointer to a palette asset if thie image is an index map instead of a color image

File

gfxu_image.h

C

```
typedef struct GFXU_ImageAsset_t {
  GFXU_AssetHeader header;
  GFXU_ImageFormat format;
  uint32_t width;
  uint32_t height;
  uint32_t bufferWidth;
  uint32_t bufferHeight;
  GFX_ColorMode colorMode;
  GFXU_ImageCompressionType compType;
  GFXU_ImageFlags flags;
  GFX_Color mask;
  GFXU_PaletteAsset* palette;
} GFXU_ImageAsset;
```

Description

Structure: GFXU_ImageAsset_t

GFXU_ImageCompressionType Enumeration

Indicates the image compression type, only applies to RAW types

File

gfxu_image.h

C

```
typedef enum GFXU_ImageCompressionType_t {
  GFXU_IMAGE_COMPRESSION_NONE = 0,
  GFXU_IMAGE_COMPRESSION_RLE
} GFXU_ImageCompressionType;
```

Description

Enumeration: GFXU_ImageCompressionType_t

GFXU_ImageFormat Enumeration

Indicates the image encoding format

File

gfxu_image.h

C

```
typedef enum GFXU_ImageFormat_t {
  GFXU_IMAGE_FORMAT_RAW = 0,
  GFXU_IMAGE_FORMAT_JPEG,
  GFXU_IMAGE_FORMAT_PNG
} GFXU_ImageFormat;
```

Description

Enumeration: GFXU_ImageFormat_t

GFXU_MemoryIntf Structure

Defines a memory interface for all memory operations. Essentially wraps a GFX_MemoryIntf with the notable addition If a GFXU_MemoryReadRequest_FnPtr.

The 'read' function pointer will be invoked when a decoder attempts to access an asset that has a location that is greater than zero. The application must then look at the asset's location id and determine which peripheral to communicate with.

File

gfxu_global.h

C

```
typedef struct GFXU_MemoryIntf_t {
  GFX_MemoryIntf heap;
  GFXU_MediaOpenRequest_FnPtr open;
  GFXU_MediaReadRequest_FnPtr read;
  GFXU_MediaCloseRequest_FnPtr close;
} GFXU_MemoryIntf;
```

Description

Structure: GFXU_MemoryIntf_t

heap - function pointer for memory operations read - function pointer to use for memory read requests

GFXU_PaletteAsset Structure

Describes a palette asset. A palette is a lookup table for unique colors. Given in an index, a color can be retrieved.

header - standard asset header colorCount - the number of colors contained in the palette colorMode - the color mode of the image

File

gfxu_palette.h

C

```
typedef struct GFXU_PaletteAsset_t {
  GFXU_AssetHeader header;
  uint32_t colorCount;
  GFX_ColorMode colorMode;
} GFXU_PaletteAsset;
```

Description

Structure: GFXU_PaletteAsset_t

GFXU_StringEncodingMode Enumeration

Indicates the string encoding mode type. Any characters above 255 must use UTF8 or UTF16

File

gfxu_string.h

C

```
typedef enum GFXU_StringEncodingMode_t {
  GFXU_STRING_ENCODING_ASCII,
  GFXU_STRING_ENCODING_UTF8,
  GFXU_STRING_ENCODING_UTF16
} GFXU_StringEncodingMode;
```

Description

Enumeration: GFXU_StringEncodingMode_t

GFXU_ASSET_LOCATION_INTERNAL Macro

File

gfxu_global.h

C

```
#define GFXU_ASSET_LOCATION_INTERNAL 0
```

Description

This is macro GFXU_ASSET_LOCATION_INTERNAL.

GFXU_STRING_ARRAY_SIZE Macro

File

gfxu_string.h

C

```
#define GFXU_STRING_ARRAY_SIZE 4
```

Description

defines meta data sizes for the string table, don't change!

GFXU_STRING_ENTRY_SIZE Macro

File

gfxu_string.h

C

```
#define GFXU_STRING_ENTRY_SIZE 2
```

Description

This is macro GFXU_STRING_ENTRY_SIZE.

GFXU_STRING_MAX_CHAR_WIDTH Macro

File

gfxu_string.h

C

```
#define GFXU_STRING_MAX_CHAR_WIDTH 6
```

Description

This is macro GFXU_STRING_MAX_CHAR_WIDTH.

GFXU_MediaCloseRequest_FnPtr Type

A callback that indicates that a media decoder is finished with a given media location and that the application can close if it.

The argument is the asset that was being read.

File

gfxu_global.h

C

```
typedef void (* GFXU_MediaCloseRequest_FnPtr)(GFXU_AssetHeader* ast);
```

Description

typedef: GFXU_MediaCloseRequest_FnPtr

GFXU_MediaOpenRequest_FnPtr Type

A callback that indicates that a media decoder wishes to read from an external media source and that the application should prepare that source.

The argument is the asset that needs to be read.

If the result is false then the decoder will abort.

File

gfxu_global.h

C

```
typedef GFX_Result (* GFXU_MediaOpenRequest_FnPtr)(GFXU_AssetHeader* ast);
```

Description

typedef: GFXU_MediaOpenRequest_FnPtr

GFXU_MediaReadRequest_FnPtr Type

File

gfxu_global.h

C

```
typedef GFX_Result (* GFXU_MediaReadRequest_FnPtr)(GFXU_ExternalAssetReader* reader, GFXU_AssetHeader*
asset, void*, uint32_t, uint8_t*, GFXU_MediaReadRequestCallback_FnPtr);
```

Description

callback

GFXU_MediaReadRequestCallback_FnPtr Type

A callback that signifies that a media read request has been fulfilled. Often signals a state machine to continue processing a decode operation. The argument is the reader that requested the memory.

File

gfxu_global.h

C

```
typedef void (* GFXU_MediaReadRequestCallback_FnPtr)(GFXU_ExternalAssetReader*);
```

Description

typedef: GFXU_MediaReadRequestCallback_FnPtr

Files

Files

Name	Description
gfxu_binary.h	Defines binary asset type.
gfxu_font.h	Describes font assets
gfxu_global.h	Global defines for graphics utility library.
gfxu_image.h	Defines image assets
gfxu_image_utils.h	Image return utilities
gfxu_palette.h	Defines palette assets
gfxu_string.h	Defines string table struct, string / character functions
gfxu_string_utils.h	Contains definitions for various internal string utility functions.

Description

gfxu_binary.h

Defines binary asset type.

Structures

	Name	Description
	GFXU_BinaryAsset_t	A binary asset type. Generic data that can be of any type

	GFXU_BinaryAsset	A binary asset type. Generic data that can be of any type

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Type definitions.

File Name

gfxu_binary.h

Company

Microchip Technology Inc.

gfxu_font.h

Describes font assets

Enumerations

	Name	Description
	GFXU_FontAssetBPP	Indicates the bits per pixel mode of a font

Structures

	Name	Description
	GFXU_FontAsset_t	Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data. header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable -... more
	GFXU_FontGlyphIndexTable_t	Describes a font glyph index table. Essentially a series of glyph look up tables where each non-consecutive range of glyphs occupies one range entry count - number of ranges in the index table ranges - the glyph range array
	GFXU_FontGlyphRange_t	Describes a range of glyphs for a font. All IDs are in raw code points and are not encoded in any way glyphCount - number of glyphs in the range startID - the starting glyph id endID - the ending glyph id lookupTable - the corresponding look up table to find the glyph raster data
	GFXU_FontGlyphIndexTable	Describes a font glyph index table. Essentially a series of glyph look up tables where each non-consecutive range of glyphs occupies one range entry count - number of ranges in the index table ranges - the glyph range array
	GFXU_FontGlyphRange	Describes a range of glyphs for a font. All IDs are in raw code points and are not encoded in any way glyphCount - number of glyphs in the range startID - the starting glyph id endID - the ending glyph id lookupTable - the corresponding look up table to find the glyph raster data

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Type definitions.

File Name

gfx_font.h

Company

Microchip Technology Inc.

gfxu_global.h

Global defines for graphics utility library.

Enumerations

	Name	Description
	GFXU_AssetType_t	Enumerates known asset types.

	GFXU_ExternalAssetReaderStatus_t	Enumerates external reader state machine states. Invalid - state machine hasn't been initialized Read - state machine is ready to begin processing but hasn't been run yet Waiting - state machine is waiting for a memory operation to complete Drawing - state machine is waiting for a drawing operation to complete Finished - state machine has finished drawing the entire asset Aborted - state machine encountered an error and has aborted
	GFXU_AssetType	Enumerates known asset types.
	GFXU_ExternalAssetReaderStatus	Enumerates external reader state machine states. Invalid - state machine hasn't been initialized Read - state machine is ready to begin processing but hasn't been run yet Waiting - state machine is waiting for a memory operation to complete Drawing - state machine is waiting for a drawing operation to complete Finished - state machine has finished drawing the entire asset Aborted - state machine encountered an error and has aborted

Macros

	Name	Description
	GFXU_ASSET_LOCATION_INTERNAL	This is macro GFXU_ASSET_LOCATION_INTERNAL.

Structures

	Name	Description
	GFXU_AssetHeader_t	Defines a common header for all assets supported by the generic decoder interface.
	GFXU_ExternalAssetReader_t	Defines the base description of an external asset reader. Specific decoder implementations will build on this foundation for each decoder type.
	GFXU_MemoryIntf_t	Defines a memory interface for all memory operations. Essentially wraps a GFX_MemoryIntf with the notable addition If a GFXU_MemoryReadRequest_FnPtr. The 'read' function pointer will be invoked when a decoder attempts to access an asset that has a location that is greater than zero. The application must then look at the asset's location id and determine which peripheral to communicate with.
	GFXU_AssetHeader	Defines a common header for all assets supported by the generic decoder interface.
	GFXU_ExternalAssetReader	Defines the base description of an external asset reader. Specific decoder implementations will build on this foundation for each decoder type.
	GFXU_MemoryIntf	Defines a memory interface for all memory operations. Essentially wraps a GFX_MemoryIntf with the notable addition If a GFXU_MemoryReadRequest_FnPtr. The 'read' function pointer will be invoked when a decoder attempts to access an asset that has a location that is greater than zero. The application must then look at the asset's location id and determine which peripheral to communicate with.

Types

	Name	Description
	GFXU_ExternalAssetReaderRun_FnPtr	This function pointer represents a function that maintains the state of an external reader state machine. The argument is the state machine to process.
	GFXU_MediaCloseRequest_FnPtr	A callback that indicates that a media decoder is finished with a given media location and that the application can close if it. The argument is the asset that was being read.
	GFXU_MediaOpenRequest_FnPtr	A callback that indicates that a media decoder wishes to read from an external media source and that the application should prepare that source. The argument is the asset that needs to be read. If the result is false then the decoder will abort.
	GFXU_MediaReadRequest_FnPtr	callback
	GFXU_MediaReadRequestCallback_FnPtr	A callback that signifies that a media read request has been fulfilled. Often signals a state machine to continue processing a decode operation. The argument is the reader that requested the memory.

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Type definitions.

File Name

gfxu_global.h

Company

Microchip Technology Inc.

gfxu_image.h

Defines image assets

Enumerations

	Name	Description
	GFXU_ImageCompressionType_t	Indicates the image compression type, only applies to RAW types
	GFXU_ImageFlags_t	A list of flags describing an image asset
	GFXU_ImageFormat_t	Indicates the image encoding format
	GFXU_ImageCompressionType	Indicates the image compression type, only applies to RAW types
	GFXU_ImageFlags	A list of flags describing an image asset
	GFXU_ImageFormat	Indicates the image encoding format

Functions

	Name	Description
	GFXU_DrawImage	Draws a portion of the given image at the specified coordinates.
	GFXU_PreprocessImage	Preprocesses an image to a specified memory address.

Structures

	Name	Description
	GFXU_ImageAsset_t	Describes an image asset. header - standard asset header format - the format of the image. this directly affects which decoder is invoked when rendering the image width - the width of the image in pixels height - the height of the image in pixels colorMode - the color mode of the image compType - the compression mode of the image useMask - indicates of the mask field is used mask - may contain a masking color for the image. blit operations may reference this value and reject image pixels that match this value palette - will contain a valid... more
	GFXU_ImageAsset	Describes an image asset. header - standard asset header format - the format of the image. this directly affects which decoder is invoked when rendering the image width - the width of the image in pixels height - the height of the image in pixels colorMode - the color mode of the image compType - the compression mode of the image useMask - indicates of the mask field is used mask - may contain a masking color for the image. blit operations may reference this value and reject image pixels that match this value palette - will contain a valid... more

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Image drawing at specified coordinates

File Name

gfx_image.h

Company

Microchip Technology Inc.

gfxu_image_utils.h

Image return utilities

Functions

	Name	Description
	convertColorAndSetDraw	internal use only
	createDefaultMemIntf	internal use only
	getDiscreteValueAtIndex	internal use only
	getOffsetFromIndexAndBPP	internal use only
	getRLEDataAtIndex	internal use only

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Internal library use only

File Name

gfx_image_utils.h

Company

Microchip Technology Inc.

gfxu_palette.h

Defines palette assets

Functions

	Name	Description
	GFXU_PaletteGetColor	Gets a color from a palette asset given an index value.

Structures

	Name	Description
	GFXU_PaletteAsset_t	Describes a palette asset. A palette is a lookup table for unique colors. Given in an index, a color can be retrieved. header - standard asset header colorCount - the number of colors contained in the palette colorMode - the color mode of the image
	GFXU_PaletteAsset	Describes a palette asset. A palette is a lookup table for unique colors. Given in an index, a color can be retrieved. header - standard asset header colorCount - the number of colors contained in the palette colorMode - the color mode of the image

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Get palette color

File Name

gfx_palette.h

Company

Microchip Technology Inc.

gfxu_string.h

Defines string table struct, string / character functions

Enumerations

	Name	Description
	GFXU_StringEncodingMode_t	Indicates the string encoding mode type. Any characters above 255 must use UTF8 or UTF16
	GFXU_StringEncodingMode	Indicates the string encoding mode type. Any characters above 255 must use UTF8 or UTF16

Functions

	Name	Description
	GFXU_CalculateCharStringWidth	Gets the width of a string buffer in pixels.
	GFXU_CalculatePartialCharStringWidth	Gets the width of a partial string buffer in pixels.
	GFXU_CalculatePartialStringWidth	Gets the partial width, starting from the left, of a string contained in the string table.
	GFXU_CalculateStringWidth	Gets the width of a string contained in the string table.
	GFXU_CompareString	Compares a string table entry and a GFXU_CHAR type string.
	GFXU_DrawCharString	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawCharStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawCharSubStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawString	Draws a string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawStringClipped	Draws a clipped string at the given coordinates. Strings are drawn from the top down.
	GFXU_DrawSubStringClipped	Draws a sub-string from a string asset in a clipped rectangle at the given coordinates. Strings are drawn from the top down.

	GFXU_ExtractString	Extracts a string from the string table into a local character buffer. The local buffer must have already been allocated. Will attempt to write 'size - 1' number of characters into the buffer with the last being a zero terminator.
	GFXU_GetCharAt	Gets a character of a string contained in the string table.
	GFXU_GetCharStringLineRect	Gets the bounding rectangle for a line in a character string.
	GFXU_GetCharWidth	Gets the width of a character for a given font.
	GFXU_GetStringHeight	Gets the height of a string contained in the string table.
	GFXU_GetStringLength	Gets the length of a string contained in a string table.
	GFXU_GetStringLineRect	Gets the bounding rectangle for a line in a string asset.
	GFXU_GetStringRect	Gets the bounding rectangle for a string contained in the string table.
	GFXU_GetStringSizeInBytes	Gets the size of a string contained in a string table, in bytes.

Macros

	Name	Description
	GFXU_STRING_ARRAY_SIZE	defines meta data sizes for the string table, don't change!
	GFXU_STRING_ENTRY_SIZE	This is macro GFXU_STRING_ENTRY_SIZE.
	GFXU_STRING_MAX_CHAR_WIDTH	This is macro GFXU_STRING_MAX_CHAR_WIDTH.

Structures

	Name	Description
	GFXU_StringTableAsset_t	Describes a string table asset. There is typically only ever one of these defined at any one time. header - standard asset header languageCount - the number of languages in the string table stringCount - the number of strings in the string table stringIndexTable - the pointer to the string index table. the string index table is a table that contains all of the unique strings defined in the string table. fontTable - the font table contains an array of pointers to all defined font assets that the string table references fontIndexTable - the font index table is a table... more
	GFXU_StringTableAsset	Describes a string table asset. There is typically only ever one of these defined at any one time. header - standard asset header languageCount - the number of languages in the string table stringCount - the number of strings in the string table stringIndexTable - the pointer to the string index table. the string index table is a table that contains all of the unique strings defined in the string table. fontTable - the font table contains an array of pointers to all defined font assets that the string table references fontIndexTable - the font index table is a table... more

Types

	Name	Description
	GFXU_CHAR	strings are defined as having 32bit characters due to the need to support international code points and unicode strings encoded using these types will not be compatible with standard library string functions like strlen or strcat
	GFXU_FontAsset	Describes a font asset. A font asset is a series of raster images that represet linguistic characters. These characters are referenced by an index called a 'code point'. This code point is 1-4 bytes in length. Code points may be encoded to save space. Fonts also contain kerning data that describes character positioning data. header - standard asset header height - font height in pixels ascent - font ascent in pixels descent - font descent in pixels baseline - font baseline in pixels bpp - font pixel size, either 1 or 8. 8 is for anti-aliased font data indexTable -... more

Description

Module for Microchip Graphics Library - Graphics Utilities Library

String statistics and drawing utilities

File Name

gfx_string.h

Company

Microchip Technology Inc.

gfxu_string_utils.h

Contains definitions for various internal string utility functions.

Functions

	Name	Description
	GFXU_DecodeAndDrawString	internal use only
	GFXU_DecodeAndDrawSubString	internal use only
	GFXU_DecodeCodePoint	internal use only
	GFXU_DecodeUTF16	internal use only
	GFXU_DecodeUTF8	internal use only
	GFXU_DrawGlyph	internal use only
	GFXU_DrawGlyphRow	internal use only
	GFXU_DrawUnknownGlyph	internal use only
	GFXU_FontGetGlyphInfo	internal use only
	GFXU_FontGetLookupTableEntry	internal use only
	GFXU_GetGlyphRowDataSize	internal use only
	GFXU_StringFontIndexLookup	internal use only
	GFXU_StringIndexLookup	internal use only
	GFXU_StringLookup	internal use only

Description

Module for Microchip Graphics Library - Graphics Utilities Library

Internal use only

File Name

gfx_string_utils.h

Company

Microchip Technology Inc.

SEGGER emWin Graphics Library

This topic is a reference for the third-party SEGGER Microcontroller GmbH & Co. KG emWin software graphics library.

Description

In addition to the standard Graphics Library, the MPLAB Harmony Integrated Software Framework also offers a third-party graphics library, emWin, from SEGGER Microcontroller GmbH & Co. KG. emWin is a software graphics library that provides an efficient, processor and LCD controller-independent Graphical User Interface (GUI) for applications that operate with a graphical LCD. This library package, which includes the binary library, headers, and utility tools, is free to use as part of development using 32-bit and 16-bit products from Microchip.

The choice of using the standard MPLAB Harmony Graphics Library and/or the SEGGER emWin Graphics Library can be made during application development through the use of the MPLAB Harmony Configurator (MHC) in MPLAB Harmony.

For details about using the emWin graphics library, refer to SEGGER emWin Graphics Library.

A detailed description of the architecture and capabilities of the SEGGER emWin Graphics Library is available on the official SEGGER website at: https://www.segger.com/emwin.html https://www.segger.com/emwin.html

Index

1

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X