MPLAB® Harmony Help - TCP-IP Stack Libraries

MPLAB Harmony Integrated Software Framework v1.11

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

TCP/IP Stack Libraries Help

This section provides information on the TCP/IP Stack Library and its modules that are available in MPLAB Harmony.

TCP/IP Stack Library Overview

This section provides an overview of the TCP/IP Stack libraries that are available in MPLAB Harmony.

Introduction

This topic provides an overview of the TCP/IP Stack in MPLAB Harmony.

Description

The MPLAB Harmony TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet, and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.

Network Metrics

This topic provides TCP/IP network metrics information for PIC32MX and PIC32MZ devices.

Description

Network Metrics for the PIC32MX

Test Setup:

- PIC32 MZ Embedded Connectivity Starter Kit
- PIC32 MX Ethernet Starter Kit II
- Starter Kit I/O Expansion Board
- Fast 100 Mbps Ethernet PICtail Plus
- Windows 7 PC
- TX buffer: 7350 bytes
- RX buffer: 4400 bytes

Iperf Bandwidth Measurements

All bandwidth numbers are in kilobits per second.

Table 1: Internal MAC Measurements for PIC32MX Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	25,963	47,652	53,650	56,682	50,191
TCP Server	29,046	45,193	50,484	52,603	45,659
UDP Client	33,795	60,997	71,105	72,049	63,298
UDP Server	36,451	78,546	86,994	88,757	80,948

Table 2: Internal MAC Measurements for PIC32MZ Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	31,722	56,929	61,361	64,479	59,742
TCP Server	28,924	41,570	44,168	44,721	42,437
UDP Client	45,534	80,116	87,043	89,377	84,044
UDP Server	53,155	90,766	95,943	95,899	94,411

Table 3: ENCx24J600 MAC Measurements for PIC32MX Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	6,023	7,225	8,211	7,633	7,373
TCP Server	5,760	7,476	8,118	8,166	7,588
UDP Client	7,035	8,826	9,159	9,148	8,949
UDP Server	7,849	9,651	10,358	10,371	9,722

Table 4: Internal MAC Measurements for PIC32MX + MIPS16 Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	19,748	29,707	32,488	33,920	30,206
TCP Server	17,369	23,294	25,945	28,018	23,421
UDP Client	23,901	34,285	38,598	51,959	36,252
UDP Server	27,720	37,914	39,186	60,143	37,401

Table 5: ENCx24J600 MAC Measurements for PIC32MX + MIPS16 Devices

Type	-O0	-O1	-O2	-O3
TCP Client	5,480	6,232	6,985	6,345
TCP Server	4,984	6,262	6,454	6,644
UDP Client	6,370	7,727	8,069	8,586
UDP Server	7,109	8,471	8,702	9,289

Commands

The following commands were used to generate the network performance numbers as measured with Iperf:

UDP/IP Client:

PIC32 Board	Personal Computer
`iperf -c <PC IP Address> -u -b 100M -i 5 -t 20`	`iperf -s -u -i 5`

UDP/IP Server:

PIC32 Board	Personal Computer
`iperf -s -u -i 5`	`iperf -c mchpboard_e -u -b 100M -i 5 -t 20`

TCP/IP Client:

PIC32 Board	Personal Computer
`iperf -c <PC IP Address> -t 10 -i 5 -l 1024 -x 100M -M 1460`	`iperf -s -i 5 -m -w 10K –N`

TCP/IP Server:

PIC32 Board	Personal Computer
`iperf -s -i 5 -x 100M`	`iperf -c mchpboard_e -t 10 -i 5 -l 1024`

Flash and RAM Usage

Provides information on Flash and RAM usage by device family.

Description

Flash and RAM Usage

Table 1 and Table 2 provide Flash and RAM usage for PIC32MX and PIC32MZ devices, respectively.

Table 1: PIC32MX Device Flash and Ram Usage

Optimization	0	0	1	1	2	2	3	3	s	s
Module	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**
ARP	12,836	136	6,992	136	7,332	136	7,548	136	6,640	136
Berkeley API	19,000	76	11,080	76	10,620	76	12,368	76	9,196	76
DDNS	7,332	630	4,688	618	4,092	618	4,636	568	3,644	618
DHCP Client	11,104	76	6,444	76	6,604	76	7,376	80	5,936	76
DHCP Server	17,816	100	9,128	100	9,124	100	10,216	100	8,552	100
DNS Client	16,724	148	9,020	148	8,960	148	11,320	148	7,844	148

DNS Server	13,192	280	7,156	280	6,528	280	7,348	284	5,948	280
FTP Server	20,548	2,009	11,376	1,989	11,360	1,989	11,628	1,993	10,728	1,989
HTTP Server	18,988	2,320	10,052	2,320	10,264	2,276	11,404	2,264	9,024	2,276
ICMP (IPv4)	2,416	28	1,260	28	1,248	28	1,272	28	1,180	28
ICMPv6 (IPv6)	12,464	28	7,320	28	7,292	28	8,012	28	6,724	28
IPv4	5,144	52	2,764	52	2,864	52	2,996	52	2,652	52
IPv6	46,756	544	28,768	544	29,304	544	41,432	544	25,712	528
NetBIOS Name Server	2,244	16	1,164	16	1,136	16	1,956	16	1,128	16
SMTP Client	7,996	262	4,976	266	4,808	266	5,964	268	4,168	266
SNTP Client	3,332	182	2,232	182	2,256	182	2,288	184	2,064	182
TCP	37,736	48	20,096	48	21,096	56	24,088	60	18,524	56
Announce Support	3,488	44	1,972	44	2,004	44	2,168	44	1,916	44
Reboot Server	756	26	412	26	436	26	436	26	420	26
UDP	22,984	48	11,748	48	12,556	48	14,588	56	10,780	48
Zeroconf Local Link	188	5	140	5	152	5	152	5	132	5
Zeroconf MDNS (Bonjour Service)	4,764	24	2,600	24	2,628	24	2,688	24	2,428	24
Stack Core Library	63,368	1,776	31,168	1,768	32,848	1,768	41,112	1,768	28,132	1,768
File System Wrappers	684	16	424	16	420	16	656	16	372	16
Ethernet MAC Driver	23,412	524	12,924	524	12,756	524	14,536	524	11,576	524
Ethernet PHY Driver	17,820	296	9,580	296	10,804	296	10,924	280	9,904	296
wolfSSL Library	237,940	25,596	130,040	25,300	129,024	28,868	178,844	28,828	110,772	28,868

Table 2: PIC32MZ Device Flash and Ram Usage

Optimization	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**s**	**s**
Module	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**
ARP	12,876	168	6,920	168	7,328	168	7,580	168	6,640	168
Berkeley API	19,016	76	11,028	76	10,600	76	12,372	76	9,196	76
DDNS	7,332	630	4,676	618	4,024	618	4,628	568	3,636	618
DHCP Client	11,164	120	6,448	120	6,528	120	7,512	124	6,020	120
DHCP Server	17,980	248	9,120	248	9,200	212	10,212	212	8,312	212
DNS Client	16,752	148	8,932	148	8,944	148	11,352	148	7,912	148
DNS Server	13,224	280	7,020	280	6,640	280	7,556	284	6,000	280
FTP Server	20,676	2,009	11,384	1,989	11,336	1,989	11,684	1,993	10,720	1,989
HTTP Server	19,164	2,516	10,144	2,516	10,340	2,472	11,472	2,460	9,112	2,472
ICMP (IPv4)	2,416	28	1,252	28	1,248	28	1,272	28	1,188	28
ICMPv6 (IPv6)	12,464	28	7,300	28	7,108	28	7,960	28	6,744	28
IPv4	5,176	100	2,772	100	2,892	100	3,024	100	2,720	100
IPv6	46,836	544	28,544	544	29,300	544	41,740	544	25,680	528
NetBIOS Name Server	2,252	16	1,168	16	1,128	16	1,956	16	1,128	16
SMTP Client	7,996	262	4,908	266	4,784	266	5,944	268	4,136	266
SNTP Client	3,332	182	2,208	182	2,256	182	2,284	184	2,072	182
TCP	37,928	116	19,904	116	21,016	124	24,096	128	18,652	124
Announce Support	3,488	44	1,964	44	2,004	44	2,144	44	1,892	44
Reboot Server	788	58	460	58	476	58	476	58	460	58
UDP	23,116	144	11,704	144	12,524	144	14,600	152	10,920	144
Zeroconf Local Link	4,836	120	2,664	120	2,704	120	152	5	128	5

Zeroconf MDNS (Bonjour Service)	16,812	132	9,368	132	9,628	132	2,760	120	2,492	120
Stack Core Library	65,400	2,716	32,064	2,708	34,304	2,708	42,308	2,704	29,180	2,704
File System Wrappers	684	16	400	16	420	16	668	4	140	4
Ethernet MAC Driver	23,736	604	13,096	604	13,264	604	8,288	600	6,796	600
Ethernet PHY Driver	17,824	300	9,552	300	10,868	300	11,116	280	9,936	300
wolfSSL Library (see **Note**)	237,940	25,596	130,424	25,300	129,276	28,868	176,976	28,880	110,796	28,868

Note: wolfSSL is built with the following features:

- SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2
- Ciphers: AES, ARC4, DES, DH, DSA, HMAC, rabbit, RSA
- Hashes: md4, md5, sha, sha256

Getting Help

This topic provides information for requesting support assistance with the TCP/IP Stack.

Description

The TCP/IP Stack is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at http://support.microchip.com.

The Microchip forums are also an excellent source of information, with a very lively community dedicated specifically to Ethernet and TCP/IP discussions at http://forum.microchip.com.

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit http://techtrain.microchip.com/rtcv2/.

Building the Library

This section lists the files that are available in the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	File that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically loaded into the MPLAB X IDE project by the MHC.

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/common/big_int.c`	Library for integers greater than 32 bits.
`/src/common/hashes.c`	Utility file for MD5 and SHA-1 hash generation.
`/src/common/helpers.c`	Helper functions for string manipulation and binary to string.
`/src/common/lfsr.c`	Simple Linear Feedback Shift Register implementation.
`/src/hash_fnv.c`	FNV Hash routines
`/src/ohash.c`	Hash Table manipulation routines.
`/src/tcpip_heap_alloc.c`	Memory heap management for the TCP/IP Stack Library.
`/src/tcpip_helpers.c`	Network string helper functions (i.e., IP address string to IP).

/src/tcpip_managers.c	Central manager for TCP/IP Stack Library.
/src/tcpip_notify.c	Functions to handle the notification heap.
/src/tcpip_packet.c	Functions for managing the preallocated packet memory.
/src/system/system_debug.c	Temporary implementation of the system debug server.
/src/system/drivers/db_appio.c	Temporary driver implementation for application debug input/output.
/src/system/drivers/usart.c	Temporary driver implementation for the UART.

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/system/system_commands.c	Temporary implementation of the debug command processor.

Module Dependencies

The TCP/IP Stack Library depends on the following modules:

- Ethernet MAC Driver Library (if using the Ethernet interface)
- Ethernet PHY Driver Library (if using the Ethernet interface)
- MRF24WG Wi-Fi Driver Library (if using the Wi-Fi interface)
- SPI Driver Library (if using the Wi-Fi interface)
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Random Number Generator System Service Library
- Debug System Service Library

TCP/IP Stack Library Porting Guide

This section provides porting information for the TCP/IP Stack libraries.

Introduction

This section provides information for porting from a previous version of the TCP/IP Stack to the TCP/IP Stack within MPLAB Harmony.

Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

The MPLAB Harmony TCP/IP Stack is a completely new distribution, which is included as part of the MPLAB Harmony installation. No files need to be maintained from an existing V5 TCP/IP Stack installation.

MPLAB Harmony TCP/IP Stack Key Features

This topic provides a description of some of the key features of the MPLAB Harmony TCP/IP Stack that are not available in the V5 version of the stack.

Description

Note: The PIC18 device family is not supported by the MPLAB Harmony TCP/IP Stack.

The following are the key features of the MPLAB Harmony TCP/IP Stack:

- Multiple interfaces - Ethernet and/or Wi-Fi interfaces are supported
- Dual stack with IPv4 and/or IPv6 support:
  - UDP and TCP sockets are capable of supporting either/or IPv4/IPv6 connections
- Fully dynamic:
  - Stack initialization/deinitialization
  - Up/down interface
  - Resource management
  - Module configuration
- Improved modularity and stack layout:
  - The stack uses the drivers and services provided by the MPLAB Harmony framework (e.g., the SPI driver for the Wi-Fi interface). This improvement offers better isolation, modularity, and maintainability of the stack.
- Run-time configuration through the TCP/IP console:
  - The stack uses the System Console and Debug system services
- Interrupt driven operation
- RTOS friendly, with easy RTOS integration:
  - OSAL calls are added in all stack modules for multi-threading support
- The BSD layer compatibility has been greatly improved, new dynamic socket options have been added as well as IPv6 support.
- Run time benchmarking support using standard industry tools - Iperf
- The MPLAB Harmony file system (SYS FS) is used by the HTTP, FTP, SNMP, modules, among others. This makes the stack independent of the specific file structure of a particular media and allows web page storage on a multitude of storage media.
- The stack uses the high performance MPLAB Harmony Cryptographic Library, which is a very efficient and modern implementation supporting the latest encryption and key exchanges algorithms
- The stack security has been greatly improved by adding support of the modern wolfSSL TLS libraries

All of these features bring flexibility and many new options that were not present in the previous V5 MLA TCP/IP implementation.

Please note that a direct comparison of the code size between demo applications using the V5 and the MPLAB Harmony TCP/IP stack can be misleading because of the multitude of the extra options, features, and services that are included as part of a standard Harmony distribution. Many of the features can be disabled from the build of the stack and the comparison of the results will be more realistic.

MPLAB Harmony TCP/IP Stack Structure

This topic describes the structure of the MPLAB Harmony TCP/IP Stack.

Description

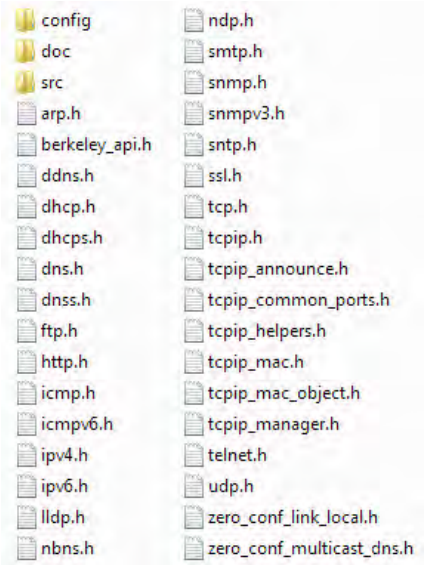
The MPLAB Harmony TCP/IP Stack consists of the following structure:

- Lowercase file names
- Underscores are used (e.g., `tcpip_manager.h`)

- Private headers are no longer exposed and have been moved to the source folder
- System services have been removed from the stack, such as:
  - Interrupts
  - Drivers
  - Timer services
  - File system
  - Board Support Package (BSP)

By default, the TCP/IP Stack is placed into the following location during installation of MPLAB Harmony on a Windows® system: `C:\Microchip\harmony\<version>\framework\tcpip`.

The following figure shows the list of files in the `tcpip` folder.



MPLAB Harmony TCP/IP Stack Design

This topic discusses the design of the MPLAB Harmony TCP/IP Stack.

Description

The MPLAB Harmony TCP/IP Stack is designed as a part of a system running other applications, middleware, etc. Therefore, it makes use of the system services that are available to other modules in the system or to the application, such as:

- File system
- System interrupts
- System driver services
- System timers
- System device drivers
- System command processor
- System console
- System debug services

Refer to the `./framework/tcpip/src/system` folder for the header files that expose the system wide available API.

MPLAB Harmony TCP/IP Stack API Changes

This topic discusses the reasons behind the API changes to the MPLAB Harmony TCP/IP Stack.

Description

For most modules, the API should be backward compatible to the V5 TCP/IP Stack.

The changes in the stack API have been minimized so that the porting process is straightforward. However, new functionality has been added because of new features, such as IPv6 support, multiple network interfaces, and the dynamic configuration of the stack. Other than these new features, the changes to the API were only made when an existing V5 TCP/IP Stack function did not support the required parameters/flexibility, or they were confusing.

The API changes are at the TCP, UDP, ARP, HTTP, SMTP, and stack access and initialization level.

TCP Changes

This topic discusses the changes made to the `tcp.h` file.

Description

TCP Changes

The TCP/IP Stack include header file, `tcp.h`, has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a TCP socket. The TCP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a socket (see `dns.h`).
- *IPv6 Support* - Support for IPv6 has been added

Opening TCP Sockets

To open TCP sockets, the required functions are now:

```
TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS* localAddress)
```

and

```
TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress)
```

These two new functions replace the V5 TCP/IP Stack TCPOpen function. The application code must replace the calls to TCPOpen with TCP_SOCKET TCPIP_TCP_ServerOpen or TCP_SOCKET TCPIP_TCP_ClientOpen, as appropriate.

The new calls add the possibility to use IPV6 addresses and make clear the choice of parameters for server or client sockets. The parameters are:

```
typedef union
{
IPV4_ADDR v4Add;
IPV6_ADDR v6Add;
}IP_MULTI_ADDRESS;
```

```
typedef enum
{
IP_ADDRESS_TYPE_ANY = 0, // either IPv4 or IPv6; unspecified;
IP_ADDRESS_TYPE_IPV4 = 1, // IPv4 address type
IP_ADDRESS_TYPE_IPV6 // IPv6 address type
}IP_ADDRESS_TYPE;
```

Note: Currently, `IP_ADDRESS_TYPE_ANY` is only supported for server sockets.

Disconnecting a Socket

The functions to disconnect a socket have changed to include added functionality:

```
TCPIP_TCP_Disconnect(TCP_SOCKET hTCP)
```

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection. It no longer sends a RST signal to the remote node so that a sequence of two sequential calls to the function is no longer needed.

If the socket has the Linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the Linger option is off, the queued TX data will be discarded.

No more data can be sent by this socket, but more data can be received (the socket will be eventually closed when a FIN is received from the remote node or by a time-out dictated by the TCP_FIN_WAIT_2_TIMEOUT value in `tcp_config.h`).

Please note that this call may fail in which case it can be reissued.

Setting the socket options is done using the function TCPIP_TCP_OptionsSet.

```
TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket)
```

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client socket will always be closed and the associated resources released. The socket cannot be used again after this call.

A server socket will abort the current connection if:

- `killSocket == false` (the socket will remain listening)
- `killSocket == true` (the socket will be closed and all associated resources released. The socket cannot be used again after this call.)

```
TCPIP_TCP_Close(TCP_SOCKET hTCP)
```

This function disconnects an open socket and destroys the socket handle, releasing the associated resources.

If the graceful option is set for the socket (default) a `TCPIP_TCP_Disconnect` will be tried (FIN will be sent):

- If the linger option is set (default) the `TCPIP_TCP_Disconnect` will attempt to send any queued TX data before issuing the FIN
- If the FIN send operation fails or the socket is not connected, an abort is generated
- If the graceful option is not set, or the previous step could not send the FIN:
- A TCPIP_TCP_Abort(is called, sending a RST to the remote node and communication is closed. The socket is no longer valid and the associated resources are freed.

Setting the socket options is done using the call TCPIP_TCP_OptionsSet();.

Adjusting Socket Size

The function to adjust the size of a socket's RX and TX buffers has changed to include added functionality:

```
TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize, TCP_ADJUST_FLAGS vFlags)
```

The TX and RX FIFOs (buffers) associated with a socket are now completely separate and independent.

Two new flags, TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY, have been added, which allow changing the size of TX and RX buffers independently. This is the preferred option.

However, when either flag is not set, for the purpose of this function, the TX and RX FIFOs are considered to be contiguous so that the total FIFO space is divided between the TX and RX FIFOs. This provides backward compatibility with previous versions of this function.

Note: The TX or RX associated buffer sizes can be independently changed too using the socket options. See the `TCPIP_TCP_OptionsSet` function for more information.

UDP Changes

This topic discusses the changes made to the `udp.h` file.

Description

UDP Changes

The TCP/IP Stack include header file, `udp.h`, has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a UDP socket. The UDP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a socket (see `dns.h`).
- *IPv6 Support* - Support for IPv6 has been added

Opening UDP Sockets

To open UDP sockets, the required functions are now:

```
UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

and

```
UDP_SOCKET TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

These two new functions replace the V5 TCP/IP Stack `UDPOpen` function.

The application code must replace any calls to UDPOpen with UDP_SOCKET TCPIP_UDP_ServerOpen or UDP_SOCKET TCPIP_UDP_ClientOpen, as appropriate.

Note: Currently, IP_ADDRESS_TYPE_ANY is only supported for server sockets.

ARP Changes

This topic discusses the changes made to the `arp.h` file.

Description

ARP Changes

The header file, `arp.h`, has been updated as follows.

The ARP module has been redesigned to support multiple network interfaces and to implement internal storage (caches) per interface for eliminating the need for frequent access to the network.

Some of the most important changes include:

- Manipulation/control of the cache entries (TCPIP_ARP_EntryGet, TCPIP_ARP_EntrySet, TCPIP_ARP_EntryRemove) with permanent entries support
- Dynamic notification mechanism (TCPIP_ARP_HandlerRegister, TCPIP_ARP_HandlerDeRegister) for signaling of the ARP events
- Resolution calls (TCPIP_ARP_Resolve, TCPIP_ARP_Probe, TCPIP_ARP_IsResolved)

Normally, the application does not need access to the ARP module. The address resolution is performed internally by the stack.

The ARP module cache manipulation access is meant for TCP/IP Stack control and configuration depending on the actual network topology.

Berkeley Changes

This topic discusses the changes made to the Berkeley API file.

Description

The Berkeley API include file has been renamed to `berkeley_api.h` (formerly `BerkeleyAPI.h`). The following changes are included:

- IPv6 Support – Support for IPv6 has been added
- setsockopt, getsockopt, gethostbyname have been added

API functionality has changed in the following ways to bring the MPLAB Harmony implementation to be more in line with industry standard Berkeley Socket API specification.

- *errno* – APIs now report specific error codes through a global variable: errno. This variable is only valid between calls to the Berkeley API, and is currently not tread safe, there is on errno variable shared across all threads.
- *bind* – bind now supports binding to a specific IP address. If an invalid address is specified the function will now return an error and set errno accordingly
- *accept* – accept no longer returns INVALID_SOCKET on error but SOCKET_ERROR and errno is set to the specific error.
- *connect* - connect no longer returns INVALID_SOCKET on error but SOCKET_ERROR and errno is set to the specific error. In addition, connect does not return SOCKET_CNXN_IN_PROGRESS; instead, SOCKET_ERROR is returned and errno is set to EINPROGRESS. Finally, connect now supports setting a specific local IP address; if the IP address is invalid, the SOCKET_ERROR is returned and errno is set with the error.
- *sendto* – sendto now supports sending from a specific IP address. If an invalid address is specified, the function returns SOCKET_ERROR and sets errno.

HTTP Changes

This topic discusses the changes made to the `http.h` file.

Note: This file was named `http2.h` in the V6 Beta TCP/IP Stack.

Description

HTTP Changes

The TCP/IP Stack include header file, `http.h`, has been updated as follows.

Function Name Changes

Prior Function Name	New Function Name
HTTPExecutePost	TCPIP_HTTP_PostExecute
HTTPExecuteGet	TCPIP_HTTP_GetExecute
HTTPNeedsAuth	TCPIP_HTTP_FileAuthenticate
HTTPCheckAuth	TCPIP_HTTP_UserAuthenticate
HTTPReadPostPair	TCPIP_HTTP_PostReadPair
HTTPPrint_varname(...)	TCPIP_HTTP_Print_varname(…)

All of the functions that the HTTP exposes as its API now take a first parameter, a HTTP connection handle, named HTTP_CONN_HANDLE. This allows a cleaner and better behavior in both multi-threaded environments and in the situation where we it may be desired to run multiple instances of the HTTP process itself, allowing for serving multiple independent connections.

For example, the HTTP functions now appear as:

- TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile) ;
- TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle);

Support has been added for the HTTP module client to store/retrieve its own connection related data. For example:

- TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData);
- TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle);

HTTP Print

Also, the functions exposed in `http_print.h` take the connection handle parameter. For example:

- TCPIP_HTTP_Print(HTTP_CONN_HANDLE connHandle, uint32_t callbackID);
- TCPIP_HTTP_Print_hellomsg(HTTP_CONN_HANDLE connHandle);

These changes affect all of the function calls in the file, `custom_http_app.c`, such as:

- TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);

To port an existing application, the extra parameter will have to be added. The connection handle is passed to the application as part of the HTTP

callback. The modifications should be minimal and possible using an editor without any other impact.

Please note that the MPFS2 generator utility (`mpfs2.jar`) has been updated to support the new HTTP API.

See the list of the complete API changes in `http.h` and `http_print.h`.

MPLAB Harmony TCP/IP Stack Storage Changes

This topic describes the storage changes in the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Storage Changes

The V5 TCP/IP Stack include header file, `tcpip_storage.h`, *does not* exist in the MPLAB Harmony TCP/IP Stack.

The TCP/IP Stack storage has become obsolete and is no longer maintained. This is mainly due to the way the dynamic initialization of the stack is done in the V5 TCP/IP Stack:

- Each module has its own initialization data
- There are many parameters that could be relevant for an application and that may require storage besides the IP address, IP Mask, or SSID, etc.
- The data is passed dynamically at the stack initialization. There is no longer a "build time" set of parameters against which to check at stack configuration (although support for a default configuration set at build time is present). The actual data contents are the application's responsibility.
- A system database service will be added that will use the File System. The database service will maintain configurations for all the modules in the system, including the TCP/IP. The system database service API will be available to the applications as well for storing/retrieving proprietary information.
- The parameter, TCPIP_STACK_USE_STORAGE, in the file, `tcpip_config.h`, should not be enabled. Future plans call for this service to be removed from the distribution.

MPLAB Harmony TCP/IP Stack Configuration Changes

This topic describes the configuration changes in the MPLAB Harmony TCP/IP Stack.

Description

The stack configuration has a completely *new* structure. Each project has its own configuration folder that stores multiple configurations based on CPU platforms and hardware boards.

Note: V5 TCP/IP Stack configuration files *cannot* be reused in the MPLAB Harmony TCP/IP Stack.

The following is a list of the most important features:

- `system_config` folder for projects – all profiles are stored in this location
- Multiple profiles per project, which are separated by CPU platform. The TCP/IP profile is usually common.
- `tcpip_config.h` – selects modules that are part of the build
- Each module has its own profile: `arp_config.h`, `tcp_config.h`, `ssl_config.h`, etc.
- BSP profiles per development board containing:
  - `hardware_config.h` – the BSP specific hardware settings
  - `media_storage_config.h` – storage media and partitioning settings
  - `sys_fs_config.h` – file system configuration and settings
  - `mpfs_config.h` – MPFS file system settings
  - `network_config.h` – configuration for every network interface: NetBIOS names, default IP addresses, etc.
  - `system_config.h` – system configuration (i.e, system tick, system console selection, system debug services, etc.)

MPLAB Harmony TCP/IP Stack Configuration

This topic provides information on the configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Configuration

The presented structure is simply a model and a different layout can be chosen. The only requirement is to use the proper path in your project so that the necessary configuration files are found at build time.

For example, depending on your exact hardware platform or development board, a path similar to the following can be added to your project path:

```
..\harmony\<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src\system_config\pic32_eth_sk_int_dyn\tcpip_profile
```

MPLAB Harmony TCP/IP Stack Heap Configuration

This topic provides information on the heap configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Heap Configuration

The MPLAB Harmony TCP/IP Stack uses dynamic memory for both initialization and run time buffers. Therefore, there are two requirements for a project containing the stack to work properly.

The first requirement is that the TCP/IP stack should be configured with enough heap space. The amount of heap used by the stack is specified by the TCPIP_STACK_DRAM_SIZE parameter in the `tcpip_config.h` file.

The value of the required TCP/IP heap for a project is application dependent. Some of the most important factors that impact the heap size are:

- Number of TCP sockets
  - By default, each TCP socket requires 512 bytes for a RX buffer and 512 bytes for a TX buffer. These parameters can be adjusted by modifying the values specified in `tcp_config.h` or dynamically through the TCPIP_TCP_OptionsSet function.
- Number of UDP sockets:
  - For each UDP socket that needs to initiate a transmission, the IP layer will have to allocate the required space (suggested by the functions, UDPv4IsTxPutReady or UDPv4IsTxPutReady
  - Once the UDP buffering will be added, each socket will have its own RX and TX buffers. These parameters can be adjusted by modifying the values specified in the file, `udp_config.h`, or dynamically through the TCPIP_UDP_OptionsSet function
- The type of Ethernet MAC that is used:
  - The PIC32MX6XX/7XX devices with a built-in 100 Mbps Ethernet controller use system memory for buffering incoming RX traffic. For sustained 100 Mbps operation, adequate RX buffer space must be provided. This parameter can be adjusted by modifying the values specified in the file, `network_config.h`, using the array, TCPIP_HOSTS_CONFIGURATION.
- If SSL is enabled, it will require additional buffering for encryption/decryption at run-time. The SSL module buffering requirement is dependent on the RSA key length specified in `ssl_config.h`.

The second requirement is that the project that includes the MPLAB Harmony TCP/IP Stack must be built with sufficient heap size to accommodate the stack (at a minimum, the project needs the TCPIP_STACK_DRAM_SIZE bytes of heap). This parameter is adjusted on the Linker tab in the project properties.

In general, for a TCP/IP project running on a PIC32MX device with an embedded Ethernet controller, at least 8 KB of heap space is needed by the stack. However, the following is implied:

- 100 Mbps traffic is not sustained
- No SSL
- Relatively few sockets

A typical value for comfortably handling 100 Mbps Ethernet traffic would be 40 KB of TCP/IP heap space.

The amount of the required heap is less for an external Ethernet MAC.

Keep in mind the following when assigning heap space:

- If there is not enough heap space the stack initialization may fail
- If there is not enough heap space some run time buffer allocation may fail and some packets transmission will have to be deferred until a later time, thus impacting the stack performance
- It is always a good idea to have a reserve, at least an extra 2 KB of heap space above the total amount of space that is used
- A very useful tool in understanding the heap allocation and how the space is distributed among the stack modules is the TCP/IP command processor

By enabling the parameter, TCPIP_STACK_DRAM_DEBUG_ENABLE, in the `tcpip_config.h` file, the stack will output debug messages when it runs out of memory. Then, using the `heapinfo` command at the TCP/IP command processor prompt will return a snapshot of the current TCP/IP heap status and can help in early detection of problems.

Optionally, enabling the parameter, TCPIP_STACK_DRAM_TRACE_ENABLE in the `tcpip_config.h` file, will instruct the TCP/IP heap allocation module to store trace information that will be displayed with the `heapinfo` command.

New MPLAB Harmony TCP/IP Stack API Functions

This topic describes some of the important new API functions in the MPLAB Harmony TCP/IP Stack.

Description

There are many new functions available that have been introduced to take advantage of new features, such as IPv6 and support of multiple interfaces. However, there is no concern for the porting process regarding the new API calls as the previous stack implementation did not support this kind of service.

A new API function should be added to the application only when the access to the new feature is needed.

Existent applications should port easily without using the new API.

For reference, the following are a few of the most important new API functions that could prove useful in the porting of your application:

- Initialization of the stack network interfaces:

  - TCPIP_STACK_NetUp()
  - TCPIP_STACK_NetDown()
- Default interface selection:
  - TCPIP_STACK_NetDefaultGet()
  - TCPIP_STACK_NetDefaultSet()
- TCP/UDP multiple interface support:
  - TCPIP_TCP_Bind()
  - TCPIP_TCP_RemoteBind()
  - TCPIP_TCP_SocketNetGet()
  - TCPIP_TCP_SocketNetSet()
  - TCPIP_UDPBind()
  - TCPIP_UDP_RemoteBind()
  - TCPIP_UDP_SocketNetGet()
  - TCPIP_UDP_SocketNetSet()

Refer to the file, `tcpip_manager.h`, for a complete list of all new network interface APIs.

Main Program Changes

This topic describes the main program changes in the MPLAB Harmony TCP/IP Stack.

Description

Main Program Changes

There are few changes that the main program loop has to take care of. A full working example is given with the file, `main_demo.c`, which is provided with the stack distribution.

The main program should call the following functions:

- SYS_Initialize() – This function is what initializes the system and runs the Board Support Package (BSP) specific code
- SYS_Tasks() – This is the function that takes care of the system tasks and must be called periodically
- TCPIP_STACK_Task() – This is the TCP/IP stack tasks function that runs all of the state machines that are part of the TCP/IP stack. This function must be called periodically. Please note the name change.

To gain access to a network interface the TCPIP_STACK_NetHandle("iname") function should be used. Examples are provided in the previous section.

MPLAB Harmony TCP/IP Stack Initialization Changes

This topic describes the initialization updates to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, `tcpip_manager.h`, has been updated as follows.

To initialize the stack the call is now:

```
TCPIP_STACK_Initialize(const TCPIP_NETWORK_CONFIG* pNetConf, int nNets, TCPIP_STACK_MODULE_CONFIG*
pModConfig, int nModules);
```

Where:

- `pNetConf` – is a pointer to an array of configurations for all the initialized network interfaces
- `nNets` – is the number of the network interfaces that the stack is to support
- `pModConfig` – is a pointer to an table storing configuration data for the TCP/IP stack modules
- `nModules` – is the number of entries in this table, (i.e., the number of initialized modules)

The default TCP/IP stack configuration is provided with the stack distribution and consists of:

- `network_config.h`: TCPIP_HOSTS_CONFIGURATION[] – an array of TCPIP_NETWORK_CONFIG entries describing the default configuration of each network interface
- `tcpip_modules_config.h`: TCPIP_STACK_MODULE_CONFIG_TBL[] – an array of TCPIP_STACK_MODULE_CONFIG entries storing the default configuration data for each module of the TCP/IP stack. This is just an aggregate of all the default configurations per module found in the module configuration header files, such as `tcp_config.h`, `udp_config.h`, `arp_config.h`, and so on.

These tables are defined by default in the configuration files. See MPLAB Harmony TCP/IP Stack Configuration Changes for details.

You can change the parameters of these structures as appropriate.

An example of the TCP/IP initialization is part of the distributed file, `main_demo.c`. This file is located in the following Windows® path:

`.\harmony<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src`

Note: The stack initialization may fail. The application code must check the result of this call to detect if the stack failed to initialize properly.

MPLAB Harmony TCP/IP Stack Access Changes

This topic describes the stack access updates that were made to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, `tcpip_manager.h`, has been updated as follows.

Direct access to the internally maintained stack structures has been removed. To access the information for a specific network interface a handle must be obtained using the interface name. For example:

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
```

or

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("MRF24W");
```

Once a handle to the interface has been obtained, different parameters of that network interface can be queried and set. For example:

- TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH) – returns the network interface address
- TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH) – returns the interface gateway address

Refer to the `tcpip_manager.h` file for a complete list of functions.

Note: The TCPIP_NET_HANDLE is an opaque data type.

The well known names of the network interfaces are currently in the file, `tcpip.h`.

The exact header file that exposes the network interfaces names may change in the future; however, the names of the supported network interfaces will be retained.

Use the file, `network_config.h`, as an example.

Refer to the file, `main_demo.c`, for an example of how to use the interface handle functions.

MPLAB Harmony TCP/IP Stack Utilities

This topic describes the MPLAB Harmony TPC/IP Stack utilities.

Description

The MPLAB Harmony TCP/IP Stack includes the following utilities:

- MPFS2 - The MPFS2 Java utility that assists the application in generating the MPFS image of the files needed by the Web server has been updated. This utility supports both the V5 and MPLAB Harmony versions of the TCP/IP stack with improved functionality. The V5 TCP/IP Stack utility can still be used; however, some minor adjustments may need to be done manually. Refer to the MPFS2 Utility section for more information.
- TCP/IP Discoverer - The TCP/IP Discoverer Java utility has been updated to support not only the MPLAB Harmony TCP/IP Stack, but also IPV6. This utility is up and running and is backward compatible with the V5 TCP/IP Stack. Refer to the TCP/IP Discoverer Utility section for more information.
- TCP/IP Configuration - The TCP/IP Configuration utility is currently not supported. The need of individually configuring TCP and UDP sockets is no longer needed, as all sockets behave similarly and they are allocated dynamically as needed.

MPLAB Harmony TCP/IP Stack SSL/RSA Usage

This topic provides information regarding the usage of SSL and RSA.

Description

The SSL design has been updated to support multiple interfaces. The same is true for the RSA engine. However, the SSL module does not currently use the Microchip Cryptographic (Crypto) Library. Until the proper Crypto Library support is added to SSL, the stack is distributed with the RSA and RC4 code embedded into the stack.

In the future, the Crypto Library will be completely removed from the TCP/IP stack, and the stack will be just a client to the Crypto Library.

Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

Porting an application from the V6 Beta TCP/IP Stack to the TCP/IP Stack within MPLAB Harmony consists of updating existing TCP/IP Stack function names in your application to the new MPLAB Harmony function names.

MPLAB Harmony TCP/IP Stack Function Name Compliance

This topic provides header file function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack.

Description

The replacement names chosen to make existing TCP/IP Stack functions compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following two MPLAB Harmony installation folders:

`.\harmony\<version>\framework\tcpip`

`.\harmony\<version>\framework\tcpip\src`

Each table shows the original and new names used. Note that other than name changes, no other modifications were made to the middleware functions.

Header File: arp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ARP_ENTRY_QUERY	TCPIP_ARP_ENTRY_QUERY
ARP_ENTRY_TYPE	TCPIP_ARP_ENTRY_TYPE
ARP_EVENT_HANDLER	TCPIP_ARP_EVENT_HANDLER
ARP_EVENT_TYPE	TCPIP_ARP_EVENT_TYPE
ARP_HANDLE	TCPIP_ARP_HANDLE
ARP_OPERATION_TYPE	TCPIP_ARP_OPERATION_TYPE
ARP_RESULT	TCPIP_ARP_RESULT
ARPCacheGetEntriesNo	TCPIP_ARP_CacheEntriesNoGet
ARPCacheSetThreshold	TCPIP_ARP_CacheThresholdSet
ARPEntryGet	TCPIP_ARP_EntryGet
ARPEntryQuery	TCPIP_ARP_EntryQuery
ARPEntryRemove	TCPIP_ARP_EntryRemove
ARPEntryRemoveAll	TCPIP_ARP_EntryRemoveAll
ARPEntryRemoveNet	TCPIP_ARP_EntryRemoveNet
ARPEntrySet	TCPIP_ARP_EntrySet
ARPDeRegisterHandler	TCPIP_ARP_HandlerDeRegister
ARPIsResolved	TCPIP_ARP_IsResolved
ARPProbe	TCPIP_ARP_Probe
ARPRegisterHandler	TCPIP_ARP_HandlerRegister
ARPResolve	TCPIP_ARP_Resolve
ARPRegisterCallbacks	TCPIP_ARP_CallbacksRegister

Header File: dhcp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPDeRegisterHandler	TCPIP_DHCP_HandlerDeRegister
DHCPRegisterHandler	TCPIP_DHCP_HandlerRegister
DHCPSetRequestTimeout	TCPIP_DHCP_RequestTimeoutSet
DHCP_EVENT_HANDLER	TCPIP_DHCP_EVENT_HANDLER
DHCP_EVENT_TYPE	TCPIP_DHCP_EVENT_TYPE
DHCP_HANDLE	TCPIP_DHCP_HANDLE
DHCPDisable	TCPIP_DHCP_Disable
DHCPEnable	TCPIP_DHCP_Enable
DHCPIsBound	TCPIP_DHCP_IsBound
DHCPIsEnabled	TCPIP_DHCP_IsEnabled
DHCPIsServerDetected	TCPIP_DHCP_IsServerDetected

Header File: dhcps.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled
DHCPServerLeaseEntryGet	TCPIP_DHCPS_LeaseEntryGet
DCHPServerGetPoolEntries	TCPIP_DHCPS_GetPoolEntries
DCHPServerRemovePoolEntries	TCPIP_DHCPS_RemovePoolEntries
DHCPServerDisable	TCPIP_DHCPS_Disable
DHCPServerEnable	TCPIP_DHCPS_Enable
DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled

Header File: ddns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DDNSForceUpdate	TCPIP_DDNS_UpdateForce
DDNSSetService	TCPIP_DDNS_ServiceSet
DDNSGetLastIP	TCPIP_DDNS_LastIPGet
DDNSGetLastStatus	TCPIP_DDNS_LastStatusGet

Header File: dns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DNSBeginUsage	TCPIP_DNS_UsageBegin
DNSEndUsage	TCPIP_DNS_UsageEnd
DNSResolve	TCPIP_DNS_Resolve
DNSIsResolved	TCPIP_DNS_IsResolved

Header File: http.h (formerly http2.h)

V6 Beta TCP/IP Stack Name (http2.h)	MPLAB Harmony Compliant Name (http.h)
HTTPURLDecode	TCPIP_HTTP_URLDecode
HTTPGetArg	TCPIP_HTTP_ArgGet
HTTPReadPostName	TCPIP_HTTP_PostNameRead
HTTPReadPostValue	TCPIP_HTTP_PostValueRead
HTTPIncFile	TCPIP_HTTP_FileInclude
HTTPCurConnectionFileGet	TCPIP_HTTP_CurrentConnectionFileGet
HTTPCurConnectionPostSmGet	TCPIP_HTTP_CurrentConnectionPostSmGet
HTTPCurConnectionPostSmSet	TCPIP_HTTP_CurrentConnectionPostSmSet
HTTPCurConnectionDataBufferGet	TCPIP_HTTP_CurrentConnectionDataBufferGet
HTTPCurConnectionCallbackPosGet	TCPIP_HTTP_CurrentConnectionCallbackPosGet
HTTPCurConnectionCallbackPosSet	TCPIP_HTTP_CurrentConnectionCallbackPosSet
HTTPCurConnectionStatusSet	TCPIP_HTTP_CurrentConnectionStatusSet
HTTPCurConnectionHasArgsSet	TCPIP_HTTP_CurrentConnectionHasArgsSet
HTTPCurConnectionByteCountGet	TCPIP_HTTP_CurrentConnectionByteCountGet
HTTPCurConnectionByteCountSet	TCPIP_HTTP_CurrentConnectionByteCountSet
HTTPCurConnectionByteCountDec	TCPIP_HTTP_CurrentConnectionByteCountDec
HTTPCurConnectionSocketGet	TCPIP_HTTP_CurrentConnectionSocketGet

Header File: icmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ICMPDeRegisterCallback	TCPIP_ICMP_CallbackDeregister
ICMPRegisterCallback	TCPIP_ICMP_CallbackRegister
ICMPSendEchoRequest	TCPIP_ICMP_EchoRequestSend

Header File: icmpv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_ICMPV6_DeRegisterCallback	TCPIP_ICMPV6_CallbackDeregister
TCPIP_ICMPV6_PutHeaderEchoRequest	TCPIP_ICMPV6_HeaderEchoRequestPut
TCPIP_ICMPV6_RegisterCallback	TCPIP_ICMPV6_CallbackRegister
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_AddressDestGet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_AddressSourceGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_AddressDestSet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_AddressSourceSet
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DAS_AddressSourceSelect
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_AddressUnicastAdd
TCPIP_IPV6_RemoveUnicastAddress	TCPIP_IPV6_AddressUnicastRemove
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
TCPIP_IPV6_RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeRegister

Header File: ipv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV6_InterfaceIsReady	Not Applicable
TCPIP_IPV6_Put	Not Applicable
TCPIP_IPV6_AddressIsSolicitedNodeMulticast	Not Applicable
TCPIP_IPV6_Flush	Not Applicable
TCPIP_IPV6_PutArrayHelper	TCPIP_IPV6_ArrayPutHelper
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DASSourceAddressSelect
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_DestAddressGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_DestAddressSet
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeregister
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
IPv6RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_FreePacket	TCPIP_IPV6_PacketFree
TCPIP_IPV6_SetPayload	TCPIP_IPV6_PayloadSet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_SourceAddressGet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_SourceAddressSet
TCPIP_IPV6_IsTxPutReady	TCPIP_IPV6_TxIsPutReady
TCPIP_IPV6_AllocateTxPacket	TCPIP_IPV6_TxPacketAllocate
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_UnicastAddressAdd
TCPIP_IPV6_GetArray	TCPIP_IPV6_ArrayGet
IPv6RemoveUnicastAddress	TCPIP_IPV6_UnicastAddressRemove

Header File: ndp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_NDP_NeighborConfirmReachability	TCPIP_NDP_NborReachConfirm

Header File: smtp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SMTPPutArray	TCPIP_SMTP_ArrayPut
SMTPFlush	TCPIP_SMTP_Flush
SMTPIsBusy	TCPIP_SMTP_IsBusy
SMTPIsPutReady	TCPIP_SMTP_IsPutReady
SMTPSendMail	TCPIP_SMTP_MailSend
SMTPPut	TCPIP_SMTP_Put
SMTPPutDone	TCPIP_SMTP_PutIsDone
SMTPPutString	TCPIP_SMTP_StringPut
SMTPBeginUsage	TCPIP_SMTP_UsageBegin
SMTPEndUsage	TCPIP_SMTP_UsageEnd

Header File: snmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNMPNotify	TCPIP_SNMP_Notify
SNMPIsNotifyReady	TCPIP_SNMP_NotifyIsReady
SNMPNotifyPrepare	TCPIP_SNMP_NotifyPrepare
SNMPGetPktProcessingDynMemStubPtrs	TCPIP_SNMP_PacketProcStubPtrsGet
SNMPGetProcessBuffData	TCPIP_SNMP_ProcessBufferDataGet
SNMPPutDataToProcessBuff	TCPIP_SNMP_DataCopyToProcessBuffer
SNMPRetrieveReadCommunity	TCPIP_SNMP_ReadCommunityGet
SNMPGetTrapTime	TCPIP_SNMP_TrapTimeGet
SNMPUdpClientGetNet	TCPIP_SNMP_ClientGetNet
SNMPRetrieveWriteCommunity	TCPIP_SNMP_WriteCommunityGet

Header File: sntp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNTPGetUTCSeconds	TCPIP_SNTP_UTCSecondsGet

Header File: tcpip_manager.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_SetNetAddress	TCPIP_STACK_NetAddressSet
TCPIP_STACK_SetNetGatewayAddress	TCPIP_STACK_NetAddressGatewaySet
TCPIP_STACK_SetNetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimarySet
TCPIP_STACK_SetNetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecondSet
TCPIP_STACK_SetNetMask	TCPIP_STACK_NetMaskSet
TCPIP_STACK_SetNetMacAddress	TCPIP_STACK_NetAddressMacSet
TCPIP_STACK_NetGatewayAddress	TCPIP_STACK_NetAddressGateway
TCPIP_STACK_NetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimary
TCPIP_STACK_NetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecond
TCPIP_STACK_NetMacAddress	TCPIP_STACK_NetAddressMac

TCPIP_STACK_NetBcastAddress	TCPIP_STACK_NetAddressBcast
TCPIP_STACK_SetNetBIOSName	TCPIP_STACK_NetBiosNameSet
TCPIP_STACK_GetDefaultNet	TCPIP_STACK_NetDefaultGet
TCPIP_STACK_SetDefaultNet	TCPIP_STACK_NetDefaultSet
TCPIP_STACK_IsNetLinked	TCPIP_STACK_NetIsLinked
TCPIP_STACK_CreateNetInfo	TCPIP_STACK_NetInfoCreate
TCPIP_STACK_IsNetUp	TCPIP_STACK_NetIsUp

Header File: tcp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIsGetReady	TCPIP_TCP_GetIsReady
TCPGetArray	TCPIP_TCP_ArrayGet
TCPPeek	TCPIP_TCP_Peek
TCPPeekArray	TCPIP_TCP_ArrayPeek
TCPGetRxFIFOFree	TCPIP_TCP_FifoRxFreeGet
TCPGetTxFIFOFull	TCPIP_TCP_FifoTxFullGet
TCPDiscard	TCPIP_TCP_Discard
TCPGet	TCPIP_TCP_Get
TCPFind	TCPIP_TCP_Find
TCPFindArray	TCPIP_TCP_ArrayFind
TCPAdjustFIFOSize	TCPIP_TCP_FifoSizeAdjust
TCPOpenServer	TCPIP_TCP_ServerOpen
TCPOpenClient	TCPIP_TCP_ClientOpen
TCPBind	TCPIP_TCP_Bind
TCPRemoteBind	TCPIP_TCP_RemoteBind
TCPSetOptions	TCPIP_TCP_OptionsSet
TCPGetOptions	TCPIP_TCP_OptionsGet
TCPIsConnected	TCPIP_TCP_IsConnected
TCPWasReset	TCPIP_TCP_WasReset
TCPDisconnect	TCPIP_TCP_Disconnect
TCPClose	TCPIP_TCP_Close
TCPGetSocketInfo	TCPIP_TCP_SocketInfoGet
TCPSocketSetNet	TCPIP_TCP_SocketNetSet
TCPSocketGetNet	TCPIP_TCP_SocketNetGet
TCPIsPutReady	TCPIP_TCP_PutIsReady
TCPPut	TCPIP_TCP_Put
TCPPutArray	TCPIP_TCP_ArrayPut
TCPPutString	TCPIP_TCP_StringPut
TCPFlush	TCPIP_TCP_Flush
TCPGetRxFIFOFull	TCPIP_TCP_FifoRxFullGet
TCPGetTxFIFOFree	TCPIP_TCP_FifoTxFreeGet
TCPAbort	TCPIP_TCP_Abort
TCPAddSSLListener	TCPIP_TCPSSL_ListenerAdd
TCPIsSSL	TCPIP_TCP_SocketIsSecuredBySSL
TCPSetDestinationIPAddress	TCPIP_TCP_DestinationIPAddressSet
TCPSetSourceIPAddress	TCPIP_TCP_SourceIPAddressSet

TCPSSLIsHandshaking	TCPIP_TCPSSL_StillHandshaking
TCpSSLMessageTransmit	TCPIP_TCPSSL_MessageTransmit
TCPSSLPutRecordHeader	TCPIP_TCPSSL_RecordHeaderPut
TCPStartSSLClient	TCPIP_TCPSSL_ClientStart
TCPStartSSLClientEx	TCPIP_TCPSSL_ClientBegin
TCPStartSSLServer	TCPIP_TCPSSL_ServerStart

Header File: tcpip_events.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_GetPendingEvents	TCPIP_STACK_EventsPendingGet
TCPIP_STACK_RegisterHandler	TCPIP_STACK_HandlerRegister
TCPIP_STACK_DeRegisterHandler	TCPIP_STACK_HandlerDeregister

Header File: ipv4.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV4_GetArray	TCPIP_IPV4_ArrayGet
TCPIP_IPV4_PutArray	TCPIP_IPV4_ArrayPut
TCPIP_IPV4_PutArrayHelper	TCPIP_IPV4_ArrayPutHelper
TCPIP_IPV4_GetDestAddress	TCPIP_IPV4_DestAddressGet
TCPIP_IPV4_SetDestAddress	TCPIP_IPV4_DestAddressSet
TCPIP_IPV4_FreePacket	TCPIP_IPV4_PacketFree
TCPIP_IPV4_SetPayload	TCPIP_IPV4_PayloadSet
TCPIP_IPV4_GetSourceAddress	TCPIP_IPV4_SourceAddressGet
TCPIP_IPV4_SetSourceAddress	TCPIP_IPV4_SourceAddressSet
TCPIP_IPV4_IsTxPutReady	TCPIP_IPV4_TxIsPutReady
TCPIP_IPV4_IsTxReady	TCPIP_IPV4_TxIsReady
TCPIP_IPV4_AllocateTxPacket	TCPIP_IPV4_TxPacketAllocate

Header File: udp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
UDPIsGetReady	TCPIP_UDP_GetIsReady
UDPSetRxOffset	TCPIP_UDP_RxOffsetSet
UDPGetArray	TCPIP_UDP_ArrayGet
UDPDiscard	TCPIP_UDP_Discard
UDPGet	TCPIP_UDP_Get
UDPOpenServer	TCPIP_UDP_ServerOpen
UDPOpenClient	TCPIP_UDP_ClientOpen
UDPBind	TCPIP_UDP_Bind
UDPRemoteBind	TCPIP_UDP_RemoteBind
UDPSetOptions	TCPIP_UDP_OptionsSet
UDPGetOptions	TCPIP_UDP_OptionsGet
UDPIsConnected	TCPIP_UDP_IsConnected
UDPIsOpened	TCPIP_UDP_IsOpened
UDPClose	TCPIP_UDP_Close
UDPGetSocketInfo	TCPIP_UDP_SocketInfoGet
UDPSocketSetNet	TCPIP_UDP_SocketNetSet

UDPSocketGetNet	TCPIP_UDP_SocketNetGet
UDPSetTxOffset	TCPIP_UDP_TxOffsetSet
UDPIsTxPutReady	TCPIP_UDP_TxPutIsReady
UDPPutArray	TCPIP_UDP_ArrayPut
UDPPutString	TCPIP_UDP_StringPut
UDPGetTxCount	TCPIP_UDP_TxCountGet
UDPFlush	TCPIP_UDP_Flush
UDPPut	TCPIP_UDP_Put
UDPSetBcastIPV4Address	TCPIP_UDP_BcastIPV4AddressSet
UDPSetSourceIPAddress	TCPIP_UDP_SourceIPAddressSet
UDPSetDestinationIPAddress	TCPIP_UDP_DestinationIPAddressSet

Development Information (Advance Information)

This *advance information* is specifically provided to assist MPLAB Harmony driver development for porting the existing Ethernet Controller Library and MAC/PHY drivers to the new platform.

Porting the MPLAB Harmony Drivers and Peripheral Library

This topic provides function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack for the purpose of porting to the MPLAB Harmony Ethernet Drivers and Ethernet Peripheral Library.

Description

Note: This content in this section is considered to be *Advance Information*.

Porting the MPLAB Harmony Ethernet Drivers and the MPLAB Harmony Ethernet Peripheral Library involves replacing the Ethernet Controller Library with calls to MPLAB Harmony equivalents. This information is listed in Table 1. Please note that function arguments were modified in some instances.

To understand how a function of the Ethernet Controller Library is different from its equivalent MPLAB Harmony function, the header of the MPLAB Harmony function should be examined. The MPLAB Harmony function header always lists the Ethernet Controller Library function(s) that the new MPLAB Harmony function replaces, including the arguments of the old function.

Table 1: TCP/IP Stack Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Name
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Table 2 shows the MPLAB Harmony Ethernet Controller Peripheral Library equivalents for functions in the MPLAB Harmony Ethernet MAC Driver and the MPLAB Harmony Ethernet PHY Driver.

The Ethernet Controller Library files are located in the following Windows® installation path:

```
C:\Program Files\Microchip\xc32\<version>\pic32-libs\peripheral\eth\source
```

Table 2: Ethernet Controller Peripheral Library Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet MAC Driver Name
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthInit	DRV_ETHMAC_LegacyInit
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet Peripheral Library Name
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt

Table 3 shows the existing and new function names for the TCP/IP MAC driver.

Please note that the following v6 Beta TCP/IP Stack files are now obsolete:

- `eth_pic32_ext_phy.c`
- `eth_pic32_ext_phy_*.c`
- `eth_pic32_int_mac.c`
- `mac_events_pic32.c`

Table 3: TCP/IP MAC Driver (tcpip_mac.h) – V6 Beta TCP/IP Stack to MPLAB Harmony TCP/IP Stack Conversion

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
MAC_ADDR	TCPIP_MAC_ADDR

MAC_ETHERNET_HEADER	TCPIP_MAC_ETHERNET_HEADER
MAC_NOTIFY_HANDLER	TCPIP_MAC_NOTIFY_HANDLER
MAC_PACKET	TCPIP_MAC_PACKET
MAC_RX_PKT_STAT	TCPIP_MAC_RX_PKT_STAT
MACEventAck	TCPIP_MAC_EventAck
MACEventClearNotifyEvents	TCPIP_MAC_EventNotifyClear
MACEventGetPending	TCPIP_MAC_EventPendingEventsGet
MACEventSetNotifyEvents	TCPIP_MAC_EventNotifyEventsSet
MACEventSetNotifyHandler	TCPIP_MAC_EventNotifyHandlerSet
MACFCEnable	TCPIP_MAC_FlowControlEnable
MACFCSetPauseValue	TCPIP_MAC_FlowControlPauseValueSet
MACFCSetRxWMark	TCPIP_MAC_FlowControlRxWMarkSet
TCPIP_MAC_FC_TYPE	TCPIP_MAC_FLOWCONTROLTYPE
MACCheckLink	TCPIP_MAC_LinkCheck
MACCalcIPBufferChecksum	TCPIP_MAC_ChecksumIPBufferCalc
MACCalcRxChecksum	TCPIP_MAC_ChecksumRxCalc
MACConnect	TCPIP_MAC_Connect
MACPowerDown	TCPIP_MAC_PowerDown
MACPowerUp	TCPIP_MAC_PowerUp
MACGetHeader	TCPIP_MAC_HeaderGet
MACDiscardRx	TCPIP_MAC_RxDiscard
MACGet	TCPIP_MAC_Get
MACGetArray	TCPIP_MAC_ArrayGet
MACGetReadPtrInRx	TCPIP_MAC_RxReadPtrGet
MACGetRxSize	TCPIP_MAC_RxSizeGet
MACSetBaseReadPtr	TCPIP_MAC_ReadPtrBaseSet
MACSetReadPtr	TCPIP_MAC_ReadPtrSet
MACSetReadPtrInRx	TCPIP_MAC_RxReadPtrSet
MACIsRxReady	TCPIP_MAC_RxIsReady
MACRxAcknowledge	TCPIP_MAC_RxAcknowledge
MACRxGetPacket	TCPIP_MAC_RxPacketGet
MACRxProfile	TCPIP_MAC_RxProfile
MACRxRegisterNotifyHandler	TCPIP_MAC_RxNotifyHandlerRegister
MACRxFilterOperation	TCPIP_MAC_RxFilterOperation
MACRxFilterPatternMode	TCPIP_MAC_RxFilterPatternMode
MACRxFilterSetHashTableEntry	TCPIP_MAC_RxFilterHashTableEntrySet
MACTxPacket	TCPIP_MAC_TxPacket
MACTxProfile	TCPIP_MAC_TxProfile
MACTxRegisterNotifyHandler	TCPIP_MAC_TxNotifyHandlerRegister
MACIsTxReady	TCPIP_MAC_TxIsReady
MACFlush	TCPIP_MAC_Flush
MACGetSslBaseAddr	TCPIP_MAC_SslBaseAddrGet
MACGetTxBaseAddr	TCPIP_MAC_TxBaseAddrGet
MACGetTxBuffSize	TCPIP_MAC_TxBuffSizeGet
MACPut	TCPIP_MAC_Put
MACPutArray	TCPIP_MAC_ArrayPut

MACPutHeader	TCPIP_MAC_HeaderPut
MACSetWritePtr	TCPIP_MAC_WritePtrSet

Announce Module

This section describes the TCP/IP Stack Library Announce module.

Introduction

TCP/IP Stack Library Announce Module for Microchip Microcontrollers

This library provides the API of the Announce Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This module will facilitate device discovery on DHCP enabled networks by broadcasting a UDP message on port 30303 whenever the local IP address changes.

Using the Library

This topic describes the basic architecture of the Announce TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_announce_config.h`

The interface to the Announce TCP/IP Stack library is defined in the `tcpip_announce_config.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Announce TCP/IP Stack library should include `tcpip.h`.

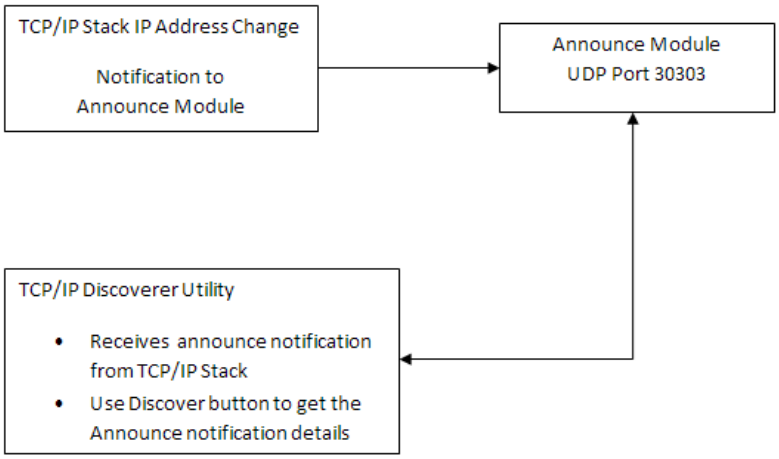
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Announce TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Announce Software Abstraction Block Diagram



Discovering the Board

Provides information on how the Announce module works in conjunction with the TCP/IP Discoverer personal computer utility to display an interface's MAC and IPv4/IPv6 addresses.

Description

The Announce module provides an easy way to determine useful parameters for your TCP/IP node's interfaces. This module works in conjunction with the TCP/IP Discoverer personal computer utility to display an interface's MAC and IPv4/IPv6 addresses, IPv6 multicast listeners, NBNS host name, and MAC type. During stack operation several events may cause an Announce packet to be broadcast on a particular interface. These include:

- Reception of an Announce packet request on that interface
- A DHCP event on that interface (i.e., the interface has configured a new IPv4 address using DHCP)
- The addition or removal of an IPv6 Unicast Address from an interface (after Duplicate Address Detection)
- The addition or removal of an IPv6 Multicast Listener from an interface
- A user call to the ANNOUNCE_Notify function specifying the interface to which you want to send parameters

Each Announce packet is a UDP packet broadcast from the corresponding interface's IPv4 address. The payload is a series of fields beginning with an 8-bit field ID, followed by field information, followed by a field terminator (0x0D 0x0A). The current field IDs and information formats are:

- 0x01 – This field ID indicates that not all of the interface's address information could be transmitted in one packet. There is no field information for this ID.
  - 0x01 0x0A 0x0D
- 0x02 – MAC address (6 bytes).
  - 0x02 0x00 0x04 0xA3 0x12 0x0f 0x94 0x0D 0x0A
- 0x03 – MAC type. The field information for this ID is a variable length string describing the MAC type of this interface – "ENCJ60", "ENCJ600", "97J60", "PIC32INT", "MRF24W".
  - 0x03 'P' 'I' 'C' '3' '2' 'I' 'N' 'T' 0x0D 0x0A
- 0x04 – Host name. The NBNS host name of the interface, with trailing spaces.
  - 0x04 'M' 'C' 'H' 'P' 'B' 'O' 'A' 'R' 'D' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' 0x0D 0x0A
- 0x05 – The interface's IPv4 address (4 bytes, big-endian).
  - 0x05 0x0A 0x00 0x01 0x03 0x0D 0x0A
- 0x06 – One of the interface's IPv6 unicast addresses (16 bytes, big-endian). Note that the interface may have more than one of these. The interface also has a corresponding solicited-node multicast address listener for every unicast address that is given. The interface may have static IP address.
  - 0x06 0xFE 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x04 0xA3 0xFF 0xFE 0x12 0x0F 0x94 0x0D 0x0A
  - 0x06 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x02 0x04 0xA3 0xFF 0xAA 0xAA 0xAA 0xAA 0x0D 0x0A
- 0x07 – One of the interface's IPv6 multicast address listeners (16 bytes, big-endian). The solicited-node multicast address listeners will not be included.
  - 0x07 0xFF 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A
- 0x08 – One of the interface's IPv6 default router address (16 bytes, big-endian).
  - 0x08 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A
- 0x09 – One of the interface's IPv6 default gateway address (16 bytes, big-endian).
  - 0x08 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A

Configuring the Library

Macros

	Name	Description
	TCPIP_ANNOUNCE_TASK_RATE	announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once

Description

The configuration of the Announce TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Announce TCP/IP Stack Library. Based on the selections made, the Announce TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Announce TCP/IP Stack 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.

TCPIP_ANNOUNCE_TASK_RATE Macro

File

tcpip_announce_config.h

C

```
#define TCPIP_ANNOUNCE_TASK_RATE 333
```

Description

announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_ANNOUNCE_MAX_PAYLOAD Macro

File

tcpip_announce_config.h

C

```
#define TCPIP_ANNOUNCE_MAX_PAYLOAD (512)
```

Description

Maximum size of a payload sent once

Building the Library

This section lists the files that are available in the Announce module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/tcpip_announce.c`	Announce 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 Announce module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Files

Files

Name	Description
tcpip_announce_config.h	Announce configuration file

Description

This section lists the source and header files used by the library.

tcpip_announce_config.h

Announce configuration file

Macros

	Name	Description
	TCPIP_ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once
	TCPIP_ANNOUNCE_TASK_RATE	announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Announce Configuration file

This file contains the Announce module configuration options

File Name

tcpip_announce_config.h

Company

Microchip Technology Inc.

ARP Module

This section describes the TCP/IP Stack Library ARP Module.

Introduction

TCP/IP Stack Library Address Resolution Protocol (ARP) Module for Microchip Microcontrollers

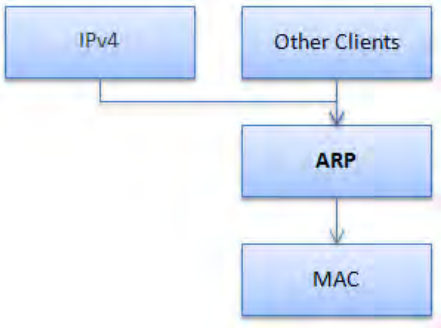
This library provides the API of the ARP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Address Resolution Protocol, or ARP, is a foundation layer of TCP/IP. It translates IP addresses to physical MAC addresses

TCP and UDP applications will not need to access ARP directly. The IPv4 module will handle the ARP operations transparently.

Responses to incoming ARP requests are processed automatically. Resolution of ARP requests follows a simple state machine, as indicated in the following diagram.



Using the Library

This topic describes the basic architecture of the ARP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `arp.h`

The interface to the arp TCP/IP Stack library is defined in the `arp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the ARP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

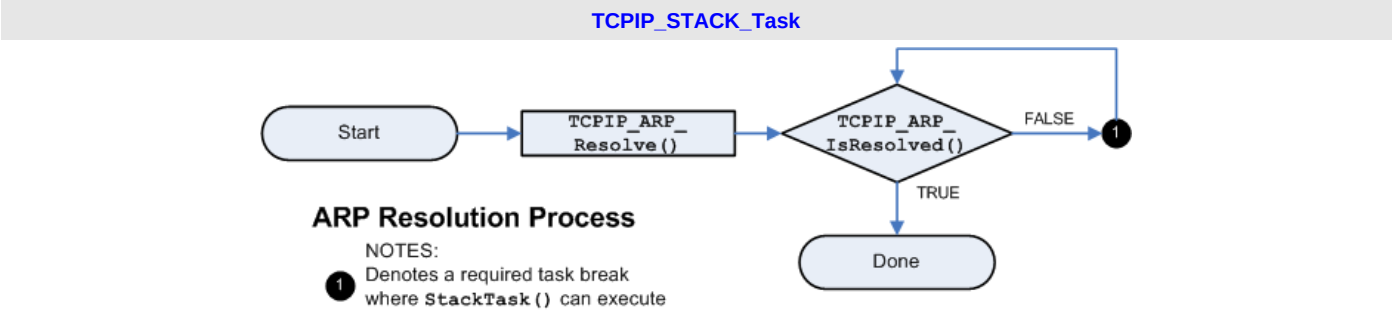
Abstraction Model

This library provides the API of the ARP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

ARP Software Abstraction Block Diagram

This module provides software abstraction of the ARP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ARP module.

Library Interface Section	Description
General Functions	Functions exported by the ARP library interface.
Cache Manipulation Functions	This section provides an interface to the internal data storage (cache) that the ARP module implements.
Data Types and Constants	Derived types and enumerations used by the ARP library interface.

How the Library Works

The ARP module provides address resolution capabilities for the TCP/IP stack.

The ARP module is an independent module that maintains its state across calls and updates its state machine. A data storage (cache) is maintained and updated internally for each interface existent in the system. The number of entries in each cache is configurable at initialization time.

The ARP state machine can remove entries from the cache and can provide signals when a specific entry has been resolved or has timed out. The purging of the cache is done internally base on a time-out parameter that is dynamically configurable.

The module provides a notification mechanism which simplifies the design of the ARP clients. The most important client of the ARP module is the IPv4 layer.

Configuring the Library

Macros

	Name	Description
	ARP_CACHE_DELETE_OLD	On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1
	TCPIP_ARP_CACHE_ENTRIES	Number of entries in the cache. Default number of entries per interface.
	TCPIP_ARP_CACHE_ENTRY_RETRIES	Number of ARP requests generated for resolving an entry.
	TCPIP_ARP_CACHE_PENDING_ENTRY_TMO	Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.
	TCPIP_ARP_CACHE_PENDING_RETRY_TMO	Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO
	TCPIP_ARP_CACHE_PERMANENT_QUOTA	Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.
	TCPIP_ARP_CACHE_PURGE_QUANTA	The number of entries to delete, once the threshold is reached.
	TCPIP_ARP_CACHE_PURGE_THRESHOLD	Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.
	TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again
	TCPIP_ARP_GRATUITOUS_PROBE_COUNT	Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.
	TCPIP_ARP_TASK_PROCESS_RATE	ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the TCP/IP Stack ARP is based on the file `system_config.h` (which may include the file `arp_config.h`).

This header file contains the configuration selection for the TCP/IP Stack ARP. Based on the selections made, the TCP/IP Stack ARP may support the selected features. These configuration settings will apply to the single instance of the TCP/IP Stack ARP.

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.

ARP_CACHE_DELETE_OLD Macro

File

arp_config.h

C

```
#define ARP_CACHE_DELETE_OLD 1
```

Description

On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1

TCPIP_ARP_CACHE_ENTRIES Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_ENTRIES 5
```

Description

Number of entries in the cache. Default number of entries per interface.

TCPIP_ARP_CACHE_ENTRY_RETRIES Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_ENTRY_RETRIES 3
```

Description

Number of ARP requests generated for resolving an entry.

TCPIP_ARP_CACHE_PENDING_ENTRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PENDING_ENTRY_TMO (1 * 60)
```

Description

Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.

TCPIP_ARP_CACHE_PENDING_RETRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PENDING_RETRY_TMO (2)
```

Description

Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO

TCPIP_ARP_CACHE_PERMANENT_QUOTA Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PERMANENT_QUOTA 50
```

Description

Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.

TCPIP_ARP_CACHE_PURGE_QUANTA Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PURGE_QUANTA 3
```

Description

The number of entries to delete, once the threshold is reached.

TCPIP_ARP_CACHE_PURGE_THRESHOLD Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PURGE_THRESHOLD 75
```

Description

Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.

TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again

TCPIP_ARP_GRATUITOUS_PROBE_COUNT Macro

File

arp_config.h

C

```
#define TCPIP_ARP_GRATUITOUS_PROBE_COUNT 1
```

Description

Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.

TCPIP_ARP_TASK_PROCESS_RATE Macro

File

arp_config.h

C

```
#define TCPIP_ARP_TASK_PROCESS_RATE (2)
```

Description

ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the ARP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/arp.c`	ARP 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 ARP module depends on the following modules:

- TCP/IP Stack Library

Library Interface

a) General Functions

	Name	Description
	TCPIP_ARP_EntryGet	Gets the current mapping for an IP address.
	TCPIP_ARP_EntryRemove	Removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_HandlerDeRegister	Deregisters the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler.

	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.
	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.
	TCPIP_ARP_Task	Standard TCP/IP stack module task function.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.

b) Cache Manipulation Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entries for a specific interface.
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in percent.
	TCPIP_ARP_EntryQuery	Queries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.

c) Data Types and Constants

	Name	Description
	TCPIP_ARP_ENTRY_QUERY	ARP entry query.
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry.
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP.
	TCPIP_ARP_HANDLE	ARP handle.
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation.
	TCPIP_ARP_RESULT	ARP results (success and failure codes).
	TCPIP_ARP_MODULE_CONFIG	This is type TCPIP_ARP_MODULE_CONFIG.

Description

This section describes the Application Programming Interface (API) functions of the ARP module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_ARP_EntryGet Function

Gets the current mapping for an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryGet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, TCPIP_MAC_ADDR* pHwAdd, bool
probe);
```

Returns

- ARP_RES_ENTRY_SOLVED - if the required entry is already solved
- ARP_RES_ENTRY_QUEUED - if the required entry was already queued
- ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
- ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
- ARP_RES_BAD_ADDRESS - bad address specified
- ARP_RES_NO_INTERFACE - no such interface

Description

If probe == false The function behaves identical to TCPIP_ARP_IsResolved(): If the corresponding MAC address exists in the cache it is copied to the user supplied pHwAdd

If probe == true The function behaves identical to TCPIP_ARP_Resolve():

- If the corresponding MAC address does not exist in the cache

this function transmits and ARP request. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

- If the hardware address exists in the cache, the result is written to pHwAdd

and no network ARP request is sent

Remarks

Similar to TCPIP_ARP_Resolve + TCPIP_ARP_IsResolved, it avoids a double hash search when the mapping exists.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The IP address to get entries from
pHwAdd	pointer to store the hardware address
probe	boolean to specify if ARP probing is initiated or not

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryGet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd,
TCPIP_MAC_ADDR* pHwAdd, bool probe)

TCPIP_ARP_EntryRemove Function

Removes the mapping of an address, even a permanent one

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemove(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_ENTRY (if no such mapping exists) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes an existent mapping from the selected interface cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	IP Address to remove entries for

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemove(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd);

TCPIP_ARP_EntryRemoveNet Function

Removes all the entries belonging to a network interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveNet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, IPV4_ADDR* mask,
TCPIP_ARP_ENTRY_TYPE type);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_BAD_TYPE (if no such type exists) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes all existent mappings belonging to a network interface. The performed operation: if(entry->type == type and entry->ipAdd & mask == ipAdd & mask) then remove entry

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
ipAdd	IP address
mask	IP address of mask
type	valid types of entries to remove: • ARP_ENTRY_TYPE_PERMANENT • ARP_ENTRY_TYPE_COMPLETE • ARP_ENTRY_TYPE_INCOMPLETE • ARP_ENTRY_TYPE_ANY

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveNet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, IPV4_ADDR* mask, TCPIP_ARP_ENTRY_TYPE type)

TCPIP_ARP_HandlerDeRegister Function

Deregisters the event handler

File

arp.h

C

```
bool TCPIP_ARP_HandlerDeRegister(TCPIP_ARP_HANDLE hArp);
```

Returns

Boolean

- On Success - true
- On Failure - false (If no such handler registered)

Description

Deregisters a previously registered ARP handler

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hArp	ARP Handle

Function

bool TCPIP_ARP_HandlerDeRegister(TCPIP_ARP_HANDLE hArp)

TCPIP_ARP_HandlerRegister Function

Register an ARP resolve handler.

File

arp.h

C

```
TCPIP_ARP_HANDLE TCPIP_ARP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_ARP_EVENT_HANDLER handler, const
void* hParam);
```

Returns

TCPIP_ARP_HANDLE

- On Success - Returns a valid handle
- On Failure - Null handle

Description

This function will register a notification handler with the ARP module.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Specifies interface to register on. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called for event.
hParam	The hParam is passed by the client and will be used by the ARP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Function

TCPIP_ARP_HANDLE TCPIP_ARP_HandlerRegister(TCPIP_NET_HANDLE hNet,
TCPIP_ARP_EVENT_HANDLER handler, const void* hParam)

TCPIP_ARP_IsResolved Function

Determines if an ARP request has been resolved yet.

File

arp.h

C

```
bool TCPIP_ARP_IsResolved(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, TCPIP_MAC_ADDR* MACAddr);
```

Description

This function checks if an ARP request has been resolved yet, and if so, stores the resolved MAC address in the pointer provided.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. This must match the IP address provided to the TCPIP_ARP_Resolve() function call.
MACAddr	A buffer to store the corresponding MAC address retrieved from the ARP query.

Function

bool TCPIP_ARP_IsResolved(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, TCPIP_MAC_ADDR* MACAddr)

TCPIP_ARP_Probe Function

Transmits an ARP probe to resolve an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_Probe(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, IPV4_ADDR* srcAddr,
TCPIP_ARP_OPERATION_TYPE opType);
```

Returns

- ARP_RES_ENTRY_SOLVED - if the required entry is already solved
- ARP_RES_ENTRY_QUEUED - if the required entry was already queued
- ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
- ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
- ARP_RES_BAD_ADDRESS - bad address specified
- ARP_RES_NO_INTERFACE - no such interface

Description

This function transmits and ARP probe to determine the hardware address of a given IP address. The packet will use the type of operation and the source address specified as parameters.

Remarks

This function is a more advanced version of TCPIP_ARP_Resolve. It allows the caller to specify the operation type and the source address of the outgoing ARP packet. It also supports the ARP flags defined in TCPIP_ARP_OPERATION_TYPE.

No check is done for IPAddr to be valid.

To retrieve the ARP query result, call the TCPIP_ARP_IsResolved function.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).
srcAddr	The source address to be used in the ARP packet
opType	Operation code to be set in the outgoing ARP packet

Function

TCPIP_ARP_RESULT TCPIP_ARP_Probe(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr,
IPV4_ADDR* srcAddr, TCPIP_ARP_OPERATION_TYPE opType)

TCPIP_ARP_Resolve Function

Transmits an ARP request to resolve an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_Resolve(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr);
```

Returns

An element from the TCPIP_ARP_RESULT enumeration. ARP_RES_ENTRY_SOLVED - if the required entry is already solved ARP_RES_ENTRY_QUEUED - if the required entry was already queued ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving) ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full ARP_RES_BAD_ADDRESS - bad address specified ARP_RES_NO_INTERFACE - no such interface

Description

This function transmits and ARP request to determine the hardware address of a given IP address. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

Remarks

To retrieve the ARP query result, call the TCPIP_ARP_IsResolved function.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).

Function

TCPIP_ARP_RESULT TCPIP_ARP_Resolve(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr)

TCPIP_ARP_Task Function

Standard TCP/IP stack module task function.

File

arp.h

C

```
void TCPIP_ARP_Task();
```

Returns

None.

Description

Performs ARP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Function

void TCPIP_ARP_Task(void)

TCPIP_ARP_EntrySet Function

Adds an ARP cache entry for the specified interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntrySet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, TCPIP_MAC_ADDR* hwAdd, bool perm);
```

Returns

- On Success - ARP_RES_OK/ARP_RES_ENTRY_EXIST
- On Failure - An Error for example, cache is full with permanent entries that cannot be purged or the permanent quota exceeded)

Description

This function will add an entry to the selected interface cache. The entry can be marked as as permanent (not subject to timeouts or updates from the network). If cache is full, an entry will be deleted to make room.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The IP address
hwAdd	The mapping MAC address for the supplied ipAdd
perm	if true, the entry will be marked as permanent

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntrySet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, TCPIP_MAC_ADDR* hwAdd, bool perm)

b) Cache Manipulation Functions

TCPIP_ARP_CacheEntriesNoGet Function

Used to retrieve the number of entries for a specific interface.

File

arp.h

C

```
size_t TCPIP_ARP_CacheEntriesNoGet(TCPIP_NET_HANDLE hNet, TCPIP_ARP_ENTRY_TYPE type);
```

Returns

The number of entries of the specified type per interface.

Description

The function will return the number of entries of the specified type that are currently in the cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
type	Type of ARP entry

Function

size_t TCPIP_ARP_CacheEntriesNoGet(TCPIP_NET_HANDLE hNet, TCPIP_ARP_ENTRY_TYPE type);

TCPIP_ARP_CacheThresholdSet Function

Sets the cache threshold for the specified interface in percent.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_CacheThresholdSet(TCPIP_NET_HANDLE hNet, int purgeThres, int purgeEntries);
```

Returns

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function sets the current value of the cache threshold for the selected interface. During the ARP operation, once the number of entries in the cache is greater than the purge threshold a number of purgeEntries (usually one) will be discarded

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
purgeThres	Threshold to start cache purging
purgeEntries	Number of entries to purge

Function

TCPIP_ARP_RESULT TCPIP_ARP_CacheThresholdSet(TCPIP_NET_HANDLE hNet,
int purgeThres, int purgeEntries);

TCPIP_ARP_EntryQuery Function

Queries an ARP cache entry using the index of the cache line.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryQuery(TCPIP_NET_HANDLE hNet, size_t index, TCPIP_ARP_ENTRY_QUERY*
pArpQuery);
```

Returns

- On Success - ARP_RES_OK
- On Failure - ARP_RES_BAD_INDEX (if index is out of range) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function can be used for displaying the cache contents.

Remarks

None.

Preconditions

The ARP module should have been initialized. The index has to be a valid one. For example, TCPIP_ARP_CacheEntriesNoGet populates the supplied TCPIP_ARP_ENTRY_QUERY query entry.

Parameters

Parameters	Description
hNet	Interface handle to use
index	Index to cache
pArpQuery	entry type, IP address, hardware address

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryQuery(TCPIP_NET_HANDLE hNet, size_t index, TCPIP_ARP_ENTRY_QUERY* pArpQuery)

TCPIP_ARP_EntryRemoveAll Function

Removes all the mapping belonging to an interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveAll(TCPIP_NET_HANDLE hNet);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes all existent mappings from the selected interface cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	network interface handle

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveAll(TCPIP_NET_HANDLE hNet)

c) Data Types and Constants

TCPIP_ARP_ENTRY_QUERY Structure

ARP entry query.

File

arp.h

C

```
typedef struct {
  TCPIP_ARP_ENTRY_TYPE entryType;
  IPV4_ADDR entryIpAdd;
  TCPIP_MAC_ADDR entryHwAdd;
} TCPIP_ARP_ENTRY_QUERY;
```

Members

Members	Description
TCPIP_ARP_ENTRY_TYPE entryType;	what entry type
IPV4_ADDR entryIpAdd;	the entry IP address
TCPIP_MAC_ADDR entryHwAdd;	the entry hardware address

Description

Structure: TCPIP_ARP_ENTRY_QUERY

Data structure for an ARP query.

TCPIP_ARP_ENTRY_TYPE Enumeration

Type of ARP entry.

File

arp.h

C

```
typedef enum {
  ARP_ENTRY_TYPE_INVALID,
  ARP_ENTRY_TYPE_PERMANENT,
  ARP_ENTRY_TYPE_COMPLETE,
  ARP_ENTRY_TYPE_INCOMPLETE,
  ARP_ENTRY_TYPE_ANY,
  ARP_ENTRY_TYPE_TOTAL
} TCPIP_ARP_ENTRY_TYPE;
```

Members

Members	Description
ARP_ENTRY_TYPE_INVALID	empty entry
ARP_ENTRY_TYPE_PERMANENT	entry valid and permanent
ARP_ENTRY_TYPE_COMPLETE	entry valid
ARP_ENTRY_TYPE_INCOMPLETE	entry not resolved yet
ARP_ENTRY_TYPE_ANY	any busy entry (PERMANENT|COMPLETE|INCOMPLETE)
ARP_ENTRY_TYPE_TOTAL	total entries - the number of entries the cache can store

Description

Enumeration: TCPIP_ARP_ENTRY_TYPE

List of different ARP cache entries.

Remarks

None.

TCPIP_ARP_EVENT_HANDLER Type

Notification handler that can be called when a specific entry is resolved.

File

arp.h

C

```
typedef void (* TCPIP_ARP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, const IPV4_ADDR* ipAdd, const
TCPIP_MAC_ADDR* MACAddr, TCPIP_ARP_EVENT_TYPE evType, const void* param);
```

Description

Type: TCPIP_ARP_EVENT_HANDLER

The format of a notification handler registered with the ARP module.

Remarks

The parameter member significance is module dependent. It can be an IP address, pointer to some other structure, etc. The handler is called

when an event of some sort occurs for a particular IP address entry. If pNetIf == 0, the notification is called for events on any interface

TCPIP_ARP_EVENT_TYPE Enumeration

Events reported by ARP.

File

arp.h

C

```
typedef enum {
  ARP_EVENT_SOLVED = 1,
  ARP_EVENT_UPDATED = 2,
  ARP_EVENT_PERM_UPDATE = 3,
  ARP_EVENT_REMOVED_TMO = -1,
  ARP_EVENT_REMOVED_USER = -2,
  ARP_EVENT_REMOVED_EXPIRED = -3,
  ARP_EVENT_REMOVED_PURGED = -4
} TCPIP_ARP_EVENT_TYPE;
```

Members

Members	Description
ARP_EVENT_SOLVED = 1	a queued cache entry was solved;
ARP_EVENT_UPDATED = 2	an existent cache entry was updated
ARP_EVENT_PERM_UPDATE = 3	an update for an permanent entry was received however the permanent entry was not updated error events - entries removed from ARP cache
ARP_EVENT_REMOVED_TMO = -1	an entry could not be solved and a tmo occurred
ARP_EVENT_REMOVED_USER = -2	a queued cache entry was removed by ARP user
ARP_EVENT_REMOVED_EXPIRED = -3	a queued cache entry expired and was removed
ARP_EVENT_REMOVED_PURGED = -4	a queued cache entry was purged

Description

Enumeration: TCPIP_ARP_EVENT_TYPE

List of events reported by ARP.

Remarks

Possibly multiple events can be set, where it makes sense.

TCPIP_ARP_HANDLE Type

ARP handle.

File

arp.h

C

```
typedef const void* TCPIP_ARP_HANDLE;
```

Description

Type: TCPIP_ARP_HANDLE

A handle that a client needs to use when deregistering a notification handler.

Remarks

This handle can be used by the client after the event handler has been registered.

TCPIP_ARP_OPERATION_TYPE Enumeration

Type of ARP operation.

File

arp.h

C

```
typedef enum {
  ARP_OPERATION_REQ = 1,
  ARP_OPERATION_RESP = 2,
  ARP_OPERATION_MASK = 0x000f,
  ARP_OPERATION_CONFIGURE = 0x1000,
  ARP_OPERATION_GRATUITOUS = 0x2000,
  ARP_OPERATION_PROBE_ONLY = 0x4000
} TCPIP_ARP_OPERATION_TYPE;
```

Members

Members	Description
ARP_OPERATION_REQ = 1	ARP request
ARP_OPERATION_RESP = 2	ARP response
ARP_OPERATION_MASK = 0x000f	extract ARP operation
ARP_OPERATION_CONFIGURE = 0x1000	stack configuration ARP packet
ARP_OPERATION_GRATUITOUS = 0x2000	stack gratuitous ARP packet
ARP_OPERATION_PROBE_ONLY = 0x4000	an ARP probe is sent only once, the target address is not stored

Description

Enumeration: TCPIP_ARP_OPERATION_TYPE

Operation to be performed by an ARP probe.

Remarks

Used for low level functionality, TCPIP_ARP_Probe.

TCPIP_ARP_RESULT Enumeration

ARP results (success and failure codes).

File

arp.h

C

```
typedef enum {
  ARP_RES_OK = 0,
  ARP_RES_ENTRY_NEW,
  ARP_RES_ENTRY_SOLVED,
  ARP_RES_ENTRY_QUEUED,
  ARP_RES_ENTRY_EXIST,
  ARP_RES_PERM_QUOTA_EXCEED,
  ARP_RES_PROBE_OK,
  ARP_RES_NO_ENTRY = -1,
  ARP_RES_CACHE_FULL = -2,
  ARP_RES_TX_FAILED = -3,
  ARP_RES_BAD_INDEX = -4,
  ARP_RES_BAD_ADDRESS = -5,
  ARP_RES_NO_INTERFACE = -6,
  ARP_RES_BAD_TYPE = -7,
  ARP_RES_CONFIGURE_ERR = -8,
  ARP_RES_PROBE_FAILED = -9
} TCPIP_ARP_RESULT;
```

Members

Members	Description
ARP_RES_OK = 0	operation succeeded
ARP_RES_ENTRY_NEW	operation succeeded and a new entry was added
ARP_RES_ENTRY_SOLVED	the required entry is already solved
ARP_RES_ENTRY_QUEUED	the required entry was already queued
ARP_RES_ENTRY_EXIST	the required entry was already cached
ARP_RES_PERM_QUOTA_EXCEED	info: the quota of permanent entries was exceeded
ARP_RES_PROBE_OK	requested probe sent
ARP_RES_NO_ENTRY = -1	no such entry exists

ARP_RES_CACHE_FULL = -2	the cache is full and no entry could be removed to make room
ARP_RES_TX_FAILED = -3	failed to transmit an ARP message
ARP_RES_BAD_INDEX = -4	bad query index
ARP_RES_BAD_ADDRESS = -5	bad IP address specified
ARP_RES_NO_INTERFACE = -6	no such interface exists
ARP_RES_BAD_TYPE = -7	no such type is valid/exists
ARP_RES_CONFIGURE_ERR = -8	interface is configuring now, no ARP probes
ARP_RES_PROBE_FAILED = -9	requested probe failed

Description

Enumeration: TCPIP_ARP_RESULT

Various definitions for success and failure codes.

Remarks

None.

TCPIP_ARP_MODULE_CONFIG Structure

File

arp.h

C

```
typedef struct {
  size_t cacheEntries;
  bool deleteOld;
  int entrySolvedTmo;
  int entryPendingTmo;
  int entryRetryTmo;
  int permQuota;
  int purgeThres;
  int purgeQuanta;
  int retries;
  int gratProbeCount;
} TCPIP_ARP_MODULE_CONFIG;
```

Members

Members	Description
size_t cacheEntries;	cache entries for this interface
bool deleteOld;	delete old cache if still in place, else don't reinitialize it
int entrySolvedTmo;	solved entry removed after this tmo if not referenced - seconds
int entryPendingTmo;	timeout for a pending to be solved entry in the cache, in seconds
int entryRetryTmo;	timeout for resending an ARP request for a pending entry - seconds 1 sec < tmo < entryPendingTmo
int permQuota;	max percentage of permanent entries allowed in the cache -
int purgeThres;	purge threshold -
int purgeQuanta;	no of entries to delete once the threshold is reached
int retries;	no of retries for resolving an entry
int gratProbeCount;	no of retries done for a gratuitous ARP request

Description

This is type TCPIP_ARP_MODULE_CONFIG.

Files

Files

Name	Description
arp.h	Address Resolution Protocol (ARP) header file.
arp_config.h	ARP configuration file

Description

This section lists the source and header files used by the library.

arp.h

Address Resolution Protocol (ARP) header file.

Enumerations

	Name	Description
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry.
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP.
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation.
	TCPIP_ARP_RESULT	ARP results (success and failure codes).

Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entries for a specific interface.
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in percent.
	TCPIP_ARP_EntryGet	Gets the current mapping for an IP address.
	TCPIP_ARP_EntryQuery	Queries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemove	Removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.
	TCPIP_ARP_HandlerDeRegister	Deregisters the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler.
	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.
	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.
	TCPIP_ARP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_ARP_ENTRY_QUERY	ARP entry query.
	TCPIP_ARP_MODULE_CONFIG	This is type TCPIP_ARP_MODULE_CONFIG.

Types

	Name	Description
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.
	TCPIP_ARP_HANDLE	ARP handle.

Description

Address Resolution Protocol (ARP) Header file

This source file contains the ARP module API definitions.

File Name

arp.h

Company

Microchip Technology Inc.

arp_config.h

ARP configuration file

Macros

	Name	Description
	ARP_CACHE_DELETE_OLD	On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1
	TCPIP_ARP_CACHE_ENTRIES	Number of entries in the cache. Default number of entries per interface.
	TCPIP_ARP_CACHE_ENTRY_RETRIES	Number of ARP requests generated for resolving an entry.
	TCPIP_ARP_CACHE_PENDING_ENTRY_TMO	Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.
	TCPIP_ARP_CACHE_PENDING_RETRY_TMO	Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO
	TCPIP_ARP_CACHE_PERMANENT_QUOTA	Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.
	TCPIP_ARP_CACHE_PURGE_QUANTA	The number of entries to delete, once the threshold is reached.
	TCPIP_ARP_CACHE_PURGE_THRESHOLD	Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.
	TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again
	TCPIP_ARP_GRATUITOUS_PROBE_COUNT	Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.
	TCPIP_ARP_TASK_PROCESS_RATE	ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Address Resolution Protocol (ARP) Configuration file

This file contains the ARP module configuration options

File Name

arp_config.h

Company

Microchip Technology Inc.

Berkeley Module

This section describes the TCP/IP Stack Library Berkeley Module.

Introduction

TCP/IP Stack Library Berkeley Module for Microchip Microcontrollers

This library provides the API of the Berkeley module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

Using the Library

This topic describes the basic architecture of the Berkeley TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `berkeley_api.h`

The interface to the Berkeley TCP/IP Stack library is defined in the `berkeley_api.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the berkeley TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

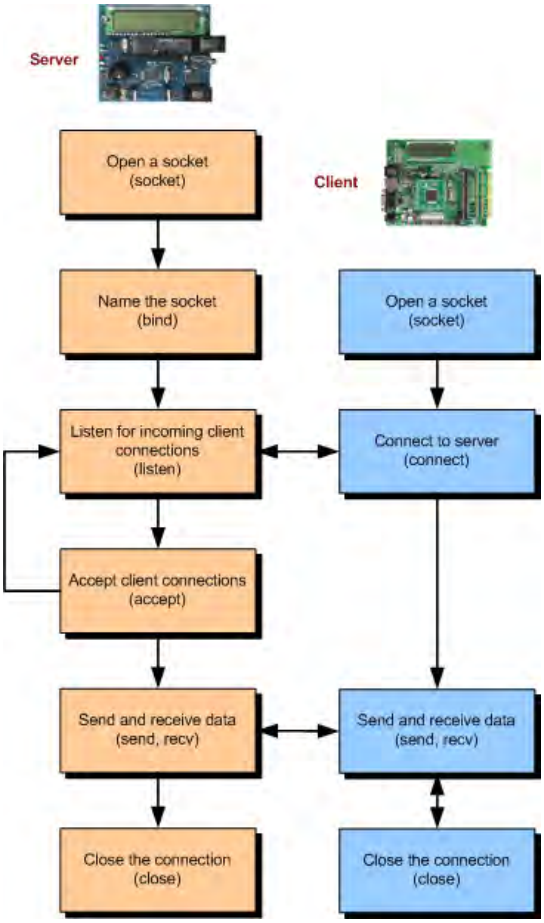
Abstraction Model

This library provides the API of the Berkeley TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Berkeley Software Abstraction Block Diagram

The following illustration shows a typical interaction for a TCP server or client using the Berkeley socket APIs.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Berkeley module.

Library Interface Section	Description
Functions	Routines for Configuring the Berkeley Socket API
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	MAX_BSD_SOCKETS	Berkeley API max number of sockets simultaneous supported

Description

The configuration of the Berkeley TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Berkeley TCP/IP Stack. Based on the selections made, the Berkeley TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the Berkeley TCP/IP Stack.

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.

MAX_BSD_SOCKETS Macro

File

berkeley_api_config.h

C

```
#define MAX_BSD_SOCKETS (4)
```

Description

Berkeley API max number of sockets simultaneous supported

Building the Library

This section lists the files that are available in the Berkeley module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/berkeley_api.c`	Berkeley API 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 Berkeley module depends on the following modules:

- TCP/IP Stack Library
- UDP Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket.
	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket.
	connect	This function connects to the peer communications end point.
	gethostname	Returns the standard host name for the system.
	listen	The listen function sets the specified socket in a listen mode.
	recv	The recv() function is used to receive incoming data that has been queued for a socket.
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket.
	send	The send function is used to send outgoing data on an already connected socket.
	sendto	This function used to send the data for both connection oriented and connection-less sockets.

	socket	This function creates a new Berkeley socket.
	getsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	setsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	gethostbyname	The gethostbyname function returns a structure of type hostent for the given host name.
	getsockname	Returns the current address to which the socket is bound.
	TCPIP_BSD_Socket	Returns the native socket number associated with the BSD socket.
	freeaddrinfo	Frees the memory allocated by getaddrinfo
	getaddrinfo	Does an address look up for the provided node name.

b) Data Types and Constants

	Name	Description
	AF_INET	Internet Address Family - IPv4, UDP, TCP, etc.
	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADDR_ANY	IP Address for server binding
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_TCP	Indicates TCP level options
	IPPROTO_UDP	Indicates UDP level options
	SOCK_DGRAM	Connectionless datagram socket. Use UDP for the Internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the Internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKET	Socket descriptor
	in_addr	in_addr structure
	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family
	addrinfo	This is record addrinfo.
	EAI_AGAIN	Temporary failure in name resolution.
	EAI_BADFLAGS	Invalid value for `ai_flags' field.
	EAI_FAIL	Non-recoverable failure in name res.
	EAI_FAMILY	ai_family' not supported.
	EAI_MEMORY	Memory allocation failure.
	EAI_NONAME	NAME or SERVICE is unknown.
	EAI_OVERFLOW	Argument buffer overflow.
	EAI_SERVICE	SERVICE not supported for `ai_socktype'.
	EAI_SOCKTYPE	ai_socktype' not supported.
	EAI_SYSTEM	System error returned in `errno'.
	ICMP6_FILTER	SEt the IPv6 Filtering options - Not yet supported
	IP_ADD_MEMBERSHIP	Join a multicast group - Not yet supported
	IP_DROP_MEMBERSHIP	Leave a multicast group - Not yet supported
	IP_MULTICAST_IF	Set the Interface that multicast packets should be sent on - Not yet supported
	IP_MULTICAST_LOOP	Specify a copy of multicast packets should be delivered to the sending host - Not yet supported
	IP_MULTICAST_TTL	Set the Time to Live option for multicast packets - Not yet supported
	IP_OPTIONS	IP Header Options - Not yet supported
	IP_TOS	Type of Service - Not yet supported
	IP_TTL	Time to Live - Not yet supported
	IPPROTO_ICMPV6	Indicates IPv6 ICMP Protocol Options
	IPPROTO_IPV6	Indicates IP v6 Header options
	IPV6_CHECKSUM	Sets the IPv6 Checksum options - Not yet supported
	IPV6_JOIN_GROUP	Join an IPv6 multicast group - Not yet supported
	IPV6_LEAVE_GROUP	Leave an IPv6 multicast group - Not yet supported

	IPV6_MULTICAST_HOPS	Set the hop limit for multicast IPv6 packets - Not yet supported
	IPV6_MULTICAST_IF	Set the Interface that multicast IPv6 packets should be sent on - Not yet supported
	IPV6_MULTICAST_LOOP	Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported
	IPV6_UNICAST_HOPS	Set the hop limit for unicast IPv6 packets - Not yet supported
	IPV6_V6ONLY	Sets the socket to IPv6 only - Not yet supported
	SO_BROADCAST	Enables the Socket for sending broadcast data
	SO_DEBUG	Indicates if low level debut is active - Not yet supported
	SO_DONTROUTE	Bypass normal routing - Not yet supported
	SO_KEEPALIVE	Keep the connection alive by sending periodic transmissions - Not yet supported
	SO_LINGER	Indicates if the system should send any buffered data when a socket is closed
	SO_OOBINLINE	Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported
	SO_RCVBUF	Receive Buffer Size (TCP only)
	SO_RCVLOWAT	Receive Low Water mark - Not yet supported
	SO_RCVTIMEO	Set the Receive Timeout - Not yet supported
	SO_REUSEADDR	Indicates if the local socket can be reused immediately after close - Not yet supported
	SO_SNDBUF	Send Buffer Size
	SO_SNDLOWAT	Send Low Water mark - Not yet supported
	SO_SNDTIMEO	Set the Send Timeout - Not yet supported
	SOL_SOCKET	Indicates socket level options
	TCP_NODELAY	Indicates if TCP is to buffer packets - Not yet supported
	linger	This is record linger.
	hostent	This is record hostent.
	HOST_NOT_FOUND	Authoritative Answer host not found
	NO_DATA	Valid name, no data record of requested type
	NO_RECOVERY	Non recoverable errors,
	TRY_AGAIN	Non authoritative host not found or server fail
	in6_addr	This is record in6_addr.
	sockaddr_in6	In the Internet address family
	sockaddr_storage	This is record sockaddr_storage.
	SOCKADDR_IN6	This is type SOCKADDR_IN6.
	AF_INET6	Internet Address Family - IPv6
	__ss_aligntype	Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.
	_BERKELEY_API_HEADER_FILE	This is macro _BERKELEY_API_HEADER_FILE.
	_SS_PADSIZE	This is macro _SS_PADSIZE.
	_SS_SIZE	This is macro _SS_SIZE.
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure

Description

This section describes the Application Programming Interface (API) functions of the Berkeley module.

Refer to each section for a detailed description.

a) Functions

accept Function

This function accepts connection requests queued for a listening socket.

File

berkeley_api.h

C

```
SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);
```

Returns

If the accept function succeeds, it returns a non-negative integer that is a descriptor for the accepted socket. Otherwise, the value SOCKET_ERROR is returned (and errno is set accordingly).

Description

The accept function is used to accept connection requests queued for a listening socket. If a connection request is pending, accept removes the request from the queue, and a new socket is created for the connection. The original listening socket remains open and continues to queue new connection requests. The socket must be a SOCK_STREAM type socket.

Remarks

None.

Preconditions

The listen function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket. Must be bound to a local name and in listening mode.
addr	Optional pointer to a buffer that receives the address of the connecting entity.
addrlen	Optional pointer to an integer that contains the length of the address addr

Function

SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen)

bind Function

This function assigns a name to the socket descriptor.

File

berkeley_api.h

C

```
int bind(SOCKET s, const struct sockaddr* name, int namelen);
```

Returns

If bind is successful, a value of 0 is returned. A return value of SOCKET_ERROR indicates an error (and errno is set accordingly).

Description

The bind function assigns a name to an unnamed socket. The name represents the local address of the communication endpoint. For sockets of type SOCK_STREAM, the name of the remote endpoint is assigned when a connect or accept function is executed.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
name	pointer to the sockaddr structure containing the local address of the socket
namelen	length of the sockaddr structure

Function

int bind(SOCKET s, const struct sockaddr* name, int namelen)

closesocket Function

The closesocket function closes an existing socket.

File

berkeley_api.h

C

```
int closesocket(SOCKET s);
```

Returns

If closesocket is successful, a value of 0 is returned. A return value of SOCKET_ERROR (-1) indicates an error (and errno is set accordingly).

Description

The closesocket function closes an existing socket. This function releases the socket descriptor s. Any data buffered at the socket is discarded. If the socket s is no longer needed, closesocket() must be called in order to release all resources associated with s.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket

Function

int closesocket(SOCKET s)

connect Function

This function connects to the peer communications end point.

File

berkeley_api.h

C

```
int connect(SOCKET s, struct sockaddr* name, int namelen);
```

Returns

If the connect() function succeeds, it returns 0. Otherwise, the value SOCKET_ERROR is returned to indicate an error condition (and errno is set accordingly). For stream based socket, if the connection is not established yet, connect returns SOCKET_ERROR and errno = EINPROGRESS.

Description

The connect function assigns the address of the peer communications endpoint. For stream sockets, connection is established between the endpoints. For datagram sockets, an address filter is established between the endpoints until changed with another connect() function.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
name	Pointer to the sockaddr structure containing the peer address and port number
namelen	Length of the sockaddr structure

Function

int connect(SOCKET s, struct sockaddr* name, int namelen)

gethostname Function

Returns the standard host name for the system.

File

berkeley_api.h

C

```
int gethostname(char* name, int namelen);
```

Returns

Success will return a value of 0. If name is too short to hold the host name or any other error occurs, SOCKET_ERROR (-1) will be returned (and errno is set accordingly). On error, *name will be unmodified and no null terminator will be generated.

Description

This function returns the standard host name of the system which is calling this function. The returned name is null-terminated.

Remarks

The function returns the host name as set on the default network interface.

Preconditions

None.

Parameters

Parameters	Description
name	Pointer to a buffer that receives the local host name
namelen	Size of the name array

Function

int gethostname(char* name, int namelen)

listen Function

The listen function sets the specified socket in a listen mode.

File

berkeley_api.h

C

```
int listen(SOCKET s, int backlog);
```

Returns

Returns 0 on success; otherwise returns SOCKET_ERROR (and errno is set accordingly).

Description

This function sets the specified socket in a listen mode. Calling the listen function indicates that the application is ready to accept connection requests arriving at a socket of type SOCK_STREAM. The connection request is queued (if possible) until accepted with an accept function. The backlog parameter defines the maximum number of pending connections that may be queued.

Remarks

None.

Preconditions

bind() must have been called on the s socket first.

Parameters

Parameters	Description
s	Socket identifier returned from a prior socket() call.
backlog	Maximum number of connection requests that can be queued. Note that each backlog requires a TCP socket to be allocated.

Function

int listen(SOCKET s, int backlog)

recv Function

The recv() function is used to receive incoming data that has been queued for a socket.

File

berkeley_api.h

C

```
int recv(SOCKET s, char* buf, int len, int flags);
```

Returns

If the recv function is successful, the socket is valid and it has pending data:

- The supplied buffer is non NULL and has non zero length, the function will return the number of bytes copied to the application buffer.
- The supplied buffer is NULL or has zero length then no data will be copied and the function will return the number of bytes pending in the socket buffer.

A return value of SOCKET_ERROR (-1) indicates an error condition (and errno set accordingly). errno is set to EWOULDBLOCK if there is no data pending in the socket buffer.

A value of zero indicates socket has been shutdown by the peer.

Description

The recv() function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream socket. If the available data is too large to fit in the supplied application buffer buf, excess bytes are discarded in case of SOCK_DGRAM type sockets. For SOCK_STREAM types, the data is buffered internally so the application can retrieve all data by multiple calls of recvfrom.

Remarks

None.

Preconditions

The connect function should be called for TCP and UDP sockets. Server side, accept function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data receive buffer
len	Buffer length in bytes
flags	No significance in this implementation

Function

int recv(SOCKET s, char* buf, int len, int flags)

recvfrom Function

The recvfrom() function is used to receive incoming data that has been queued for a socket.

File

berkeley_api.h

C

```
int recvfrom(SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen);
```

Returns

If recvfrom is successful, the number of bytes copied to the application buffer buf is returned. A return value of SOCKET_ERROR (-1) indicates an error condition (and errno is set accordingly). A value of zero indicates socket has been shutdown by the peer.

Description

The recvfrom() function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream type sockets. If the available data is too large to fit in the supplied application buffer buf, excess bytes are discarded in case of SOCK_DGRAM type sockets. For SOCK_STREAM types, the data is buffered internally so the application can retrieve all data by multiple calls of

recvfrom.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data receive buffer
len	Buffer length in bytes
flags	Message flags (currently this is not supported)
from	Pointer to the sockaddr structure that will be filled in with the destination address
fromlen	Size of buffer pointed by from

Function

int recvfrom(SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen)

send Function

The send function is used to send outgoing data on an already connected socket.

File

berkeley_api.h

C

```
int send(SOCKET s, const char* buf, int len, int flags);
```

Returns

On success, send returns number of bytes sent. Zero indicates no data send. In case of error it returns SOCKET_ERROR (and errno is set accordingly).

Description

The send function is used to send outgoing data on an already connected socket. This function is used to send a reliable, ordered stream of data bytes on a socket of type SOCK_STREAM but can also be used to send datagrams on a socket of type SOCK_DGRAM.

Remarks

None.

Preconditions

The connect function should be called for TCP and UDP sockets. Server side, accept function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data buffer containing data to transmit
len	Length of data in bytes
flags	Message flags (currently this field is not supported)

Function

int send(SOCKET s, const char* buf, int len, int flags)

sendto Function

This function used to send the data for both connection oriented and connection-less sockets.

File

berkeley_api.h

C

```
int sendto(SOCKET s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen);
```

Returns

On success, sendto returns number of bytes sent. In case of error returns SOCKET_ERROR (and errno is set accordingly).

Description

The sendto function is used to send outgoing data on a socket. The destination address is given by to and tolen. Both Datagram and stream sockets are supported.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
buf	application data buffer containing data to transmit.
len	length of data in bytes.
flags	message flags. Currently this field is not supported.
to	Optional pointer to the sockaddr structure containing the destination address. If NULL, the currently bound remote port and IP address are used as the destination.
tolen	length of the sockaddr structure.

Function

int sendto(SOCKET s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen)

socket Function

This function creates a new Berkeley socket.

File

berkeley_api.h

C

```
SOCKET socket(int af, int type, int protocol);
```

Returns

New socket descriptor. SOCKET_ERROR in case of error (and errno set accordingly).

Description

This function creates a new BSD socket for the Microchip TCP/IP stack. The return socket descriptor is used for the subsequent BSD operations.

Remarks

None.

Preconditions

BerkeleySocketInit function should be called.

Parameters

Parameters	Description
af	address family - AF_INET for IPv4, AF_INET6 for IPv6
type	socket type SOCK_DGRAM or SOCK_STREAM
protocol	IP protocol IPPROTO_UDP or IPPROTO_TCP

Function

SOCKET socket(int af, int type, int protocol)

getsockopt Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

berkeley_api.h

C

```
int getsockopt(SOCKET s, uint32_t level, uint32_t option_name, uint8_t * option_value, uint32_t *
option_length);
```

Returns

- 0 - If successful
- -1 - If unsuccessful

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
level	On which level the operation is to be performed: • IPPROTO_IP - Applies to the IP protocol layer (not yet supported) • IPPROTO_TCP - Applies to the TCP protocol layer • SOL_SOCKET - Applies to the socket layer • IPPROTO_IPV6 - Applies to the IPv6 protocol layer (not yet supported) • IPPROTO_ICMPV6 - Applies to the ICMPv6 protocol layer (not yet supported)
option_name	The name of the option to be set: • IPPROTO_TCP • TCP_NODELAY - Specifies whether or not the stack should use the Nagle algorithm • SOL_SOCKET • SO_LINGER - Specifies what the stack should do with unsent data on close() • SO_RCVBUF - Specifies the size of the Receive Buffer (TCP only) • SO_SNDBUF - Specifies the size of the Transmit Buffer
option_value	For all values of option_name this is a pointer to the data, which in most cases is an integer. The only exception is SO_LINGER which points to a linger structure.
option_length	The size of the data pointed to by option_value

Function

int getsockopt(SOCKET s, uint32_t level, uint32_t option_name, uint8_t *option_value, uint32_t *option_length)

setsockopt Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

berkeley_api.h

C

```
int setsockopt(SOCKET s, uint32_t level, uint32_t option_name, const uint8_t * option_value, uint32_t
option_length);
```

Returns

- 0 - If successful
- -1 - If unsuccessful

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
level	On which level the operation is to be performed: • IPPROTO_IP - Applies to the IP protocol layer (not yet supported) • IPPROTO_TCP - Applies to the TCP protocol layer • SOL_SOCKET - Applies to the socket layer • IPPROTO_IPV6 - Applies to the IPv6 protocol layer (not yet supported) • IPPROTO_ICMPV6 - Applies to the ICMPv6 protocol layer (not yet supported)
option_name	The name of the option to be set: • IPPROTO_TCP • TCP_NODELAY - Specifies whether or not the stack should use the Nagle algorithm • SOL_SOCKET • SO_LINGER - Specifies what the stack should do with unsent data on close() • SO_RCVBUF - Specifies the size of the Receive Buffer (TCP only) • SO_SNDBUF - Specifies the size of the Transmit Buffer
option_value	For all values of option_name this is a pointer to the data, which in most cases is an integer. The only exception is SO_LINGER which points to a linger structure.
option_length	The size of the data pointed to by option_value

Function

int setsockopt(SOCKET s, uint32_t level, uint32_t option_name, const uint8_t *option_value, uint32_t option_length)

gethostbyname Function

The gethostbyname function returns a structure of type hostent for the given host name.

File

berkeley_api.h

C

```
struct hostent * gethostbyname(char * name);
```

Returns

The hostent structure on success, or NULL on error.

Remarks

This function supports IPv4 only. h_errno will be set to:

- TRY_AGAIN if the DNS query is current in progress,
- HOST_NOT_FOUND if the DNS query could not find a host name
- NO_RECOVERY if the DNS query had an unrecoverable error

Preconditions

None.

Parameters

Parameters	Description
name	The name of the host to be found

Function

struct hostent * gethostbyname(char *name)

getsockname Function

Returns the current address to which the socket is bound.

File

berkeley_api.h

C

```
int getsockname(SOCKET s, struct sockaddr * addr, int * addrlen);
```

Returns

- On success - 0 is returned and the data is updated accordingly.
- On error - -1 is returned and errno is set appropriately.

Description

The function returns the current address to which the socket is bound, in the buffer pointed to by addr.

Remarks

This function supports IPv4 connections only.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
addr	address to store the current address to which the socket is bound
addrlen	on input it should point to the space (bytes) available in addr on output it points to the actual space required for storing the bound address

Function

int getsockname(SOCKET s, struct sockaddr *addr, int *addrlen);

TCPIP_BSD_Socket Function

Returns the native socket number associated with the BSD socket.

File

berkeley_api.h

C

```
int TCPIP_BSD_Socket(SOCKET s);
```

Returns

- On success - a socket number >= 0 is returned.
- On error - -1 is returned if no such socket exists and errno is set to EBADF.

Description

The function returns the native socket number associated with the BSD socket. Using this call the caller can switch to the native TCP/IP API.

Remarks

This function works for both TCP and UDP sockets.

The native UDP sockets are created after a call to bind (server sockets) or connect (client sockets). The native TCP sockets are created after a call to listen (server sockets) or connect (client sockets). Please note that calling the TCPIP_BSD_Socket before one of these calls will return an INVALID_SOCKET.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket

Function

int TCPIP_BSD_Socket(SOCKET s);

freeaddrinfo Function

Frees the memory allocated by getaddrinfo

File

berkeley_api.h

C

```
void freeaddrinfo(struct addrinfo * res);
```

Returns

None.

Description

Frees the memory allocated by getaddrinfo

Preconditions

getaddrinfo must have returned successfully.

Parameters

Parameters	Description
res	Memory allocated by getaddrinfo

Function

int freeaddrinfo(struct addrinfo *res);

getaddrinfo Function

Does an address look up for the provided node name.

File

berkeley_api.h

C

```
int getaddrinfo(const char * node, const char * service, const struct addrinfo * hints, struct addrinfo **
res);
```

Returns

On success, a 0 is returned.

- EAI_NONAME - When no name could be found
- EAI_FAIL - When a failure has occurred
- EAI_AGAIN - When the look-up is in progress. Call the function again later to check the results.

Description

This function deprecates gethostbyname. It handles both IPv4 and IPv6.

Preconditions

The MPLAB Harmony DNS client services must be active.

Parameters

Parameters	Description
node	The name of the server to look up.
service	Unused

hints	hints to the function. Currently only ai_family is used. Set to either 0, AF_INET or AF_INET6
res	Memory location of where the results are. Results must be freed with freeaddrinfo

Function

int getaddrinfo(const char *node, const char *service,
const struct addrinfo *hints,
struct addrinfo **res);

b) Data Types and Constants

AF_INET Macro

File

berkeley_api.h

C

```
#define AF_INET 2            // Internet Address Family - IPv4, UDP, TCP, etc.
```

Description

Internet Address Family - IPv4, UDP, TCP, etc.

INADDR_ANY Macro

File

berkeley_api.h

C

```
#define INADDR_ANY 0x01000000u    // IP address for server binding.
```

Description

IP address for server binding.

INVALID_TCP_PORT Macro

File

berkeley_api.h

C

```
#define INVALID_TCP_PORT (0L)  //Invalid TCP port
```

Description

Invalid TCP port

IP_ADDR_ANY Macro

File

berkeley_api.h

C

```
#define IP_ADDR_ANY 0x01000000u            // IP Address for server binding
```

Description

IP Address for server binding

IPPROTO_IP Macro

File

berkeley_api.h

C

```
#define IPPROTO_IP 0   // Indicates IP pseudo-protocol.
```

Description

Indicates IP pseudo-protocol.

IPPROTO_TCP Macro

File

berkeley_api.h

C

```
#define IPPROTO_TCP 6   // Indicates TCP level options
```

Description

Indicates TCP level options

IPPROTO_UDP Macro

File

berkeley_api.h

C

```
#define IPPROTO_UDP 17  // Indicates UDP level options
```

Description

Indicates UDP level options

SOCK_DGRAM Macro

File

berkeley_api.h

C

```
#define SOCK_DGRAM 110  //Connectionless datagram socket. Use UDP for the Internet address family.
```

Description

Connectionless datagram socket. Use UDP for the Internet address family.

SOCK_STREAM Macro

File

berkeley_api.h

C

```
#define SOCK_STREAM 100  //Connection based byte streams. Use TCP for the Internet address family.
```

Description

Connection based byte streams. Use TCP for the Internet address family.

SOCKET_CNXN_IN_PROGRESS Macro

File

berkeley_api.h

C

```
#define SOCKET_CNXN_IN_PROGRESS (-2) //Socket connection state.
```

Description

Socket connection state.

SOCKET_DISCONNECTED Macro

File

berkeley_api.h

C

```
#define SOCKET_DISCONNECTED (-3) //Socket disconnected
```

Description

Socket disconnected

SOCKET_ERROR Macro

File

berkeley_api.h

C

```
#define SOCKET_ERROR (-1) //Socket error
```

Description

Socket error

SOCKADDR Type

File

berkeley_api.h

C

```
typedef struct sockaddr SOCKADDR;
```

Description

generic address structure for all address families

SOCKADDR_IN Type

File

berkeley_api.h

C

```
typedef struct sockaddr_in SOCKADDR_IN;
```

Description

In the Internet address family

SOCKET Type

File

berkeley_api.h

C

```
typedef int16_t SOCKET;
```

Description

Socket descriptor

in_addr Structure

File

berkeley_api.h

C

```
struct in_addr {
  union {
    struct {
      uint8_t s_b1, s_b2, s_b3, s_b4;
    } S_un_b;
    struct {
      uint16_t s_w1, s_w2;
    } S_un_w;
    uint32_t S_addr;
  } S_un;
};
```

Members

Members	Description

union {
struct {
uint8_t s_b1, s_b2, s_b3, s_b4;
} S_un_b;
struct {
uint16_t s_w1, s_w2;
} S_un_w;
uint32_t S_addr;
} S_un;
	union of IP address

struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b;	IP address in Byte
struct { uint16_t s_w1, s_w2; } S_un_w;	IP address in Word
uint32_t S_addr;	IP address

Description

in_addr structure

sockaddr Structure

File

berkeley_api.h

C

```
struct sockaddr {
  unsigned short sa_family;
  char sa_data[14];
};
```

Members

Members	Description
unsigned short sa_family;	address family
char sa_data[14];	up to 14 bytes of direct address

Description

generic address structure for all address families

sockaddr_in Structure

File

berkeley_api.h

C

```
struct sockaddr_in {
  short sin_family;
  uint16_t sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

Members

Members	Description
short sin_family;	Address family; must be AF_INET.
uint16_t sin_port;	Internet Protocol (IP) port.
struct in_addr sin_addr;	IP address in network byte order.
char sin_zero[8];	Padding to make structure the same size as SOCKADDR.

Description

In the Internet address family

addrinfo Structure

File

berkeley_api.h

C

```
struct addrinfo {
  int ai_flags;
  int ai_family;
  int ai_socktype;
  int ai_protocol;
  size_t ai_addrlen;
  struct sockaddr * ai_addr;
  char * ai_canonname;
  struct addrinfo * ai_next;
};
```

Description

This is record addrinfo.

EAI_AGAIN Macro

File

berkeley_api.h

C

```
#define EAI_AGAIN -3    /* Temporary failure in name resolution.  */
```

Description

Temporary failure in name resolution.

EAI_BADFLAGS Macro

File

berkeley_api.h

C

```
#define EAI_BADFLAGS -1    /* Invalid value for `ai_flags' field.  */
```

Description

Invalid value for `ai_flags' field.

EAI_FAIL Macro

File

berkeley_api.h

C

```
#define EAI_FAIL -4    /* Non-recoverable failure in name res.  */
```

Description

Non-recoverable failure in name res.

EAI_FAMILY Macro

File

berkeley_api.h

C

```
#define EAI_FAMILY -6    /* `ai_family' not supported.  */
```

Description

ai_family' not supported.

EAI_MEMORY Macro

File

berkeley_api.h

C

```
#define EAI_MEMORY -10   /* Memory allocation failure.  */
```

Description

Memory allocation failure.

EAI_NONAME Macro

File

berkeley_api.h

C

```
#define EAI_NONAME -2    /* NAME or SERVICE is unknown.  */
```

Description

NAME or SERVICE is unknown.

EAI_OVERFLOW Macro

File

berkeley_api.h

C

```
#define EAI_OVERFLOW -12   /* Argument buffer overflow.  */
```

Description

Argument buffer overflow.

EAI_SERVICE Macro

File

berkeley_api.h

C

```
#define EAI_SERVICE -8    /* SERVICE not supported for `ai_socktype'.  */
```

Description

SERVICE not supported for `ai_socktype'.

EAI_SOCKTYPE Macro

File

berkeley_api.h

C

```
#define EAI_SOCKTYPE -7    /* `ai_socktype' not supported.  */
```

Description

ai_socktype' not supported.

EAI_SYSTEM Macro

File

berkeley_api.h

C

```
#define EAI_SYSTEM -11   /* System error returned in `errno'.  */
```

Description

System error returned in `errno'.

ICMP6_FILTER Macro

File

berkeley_api.h

C

```
#define ICMP6_FILTER 1    //SEt the IPv6 Filtering options - Not yet supported
```

Description

SEt the IPv6 Filtering options - Not yet supported

IP_ADD_MEMBERSHIP Macro

File

berkeley_api.h

C

```
#define IP_ADD_MEMBERSHIP 35  //Join a multicast group - Not yet supported
```

Description

Join a multicast group - Not yet supported

IP_DROP_MEMBERSHIP Macro

File

berkeley_api.h

C

```
#define IP_DROP_MEMBERSHIP 36  // Leave a multicast group - Not yet supported
```

Description

Leave a multicast group - Not yet supported

IP_MULTICAST_IF Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_IF 32  //Set the Interface that multicast packets should be sent on  - Not yet
supported
```

Description

Set the Interface that multicast packets should be sent on - Not yet supported

IP_MULTICAST_LOOP Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_LOOP 34  //Specify a copy of multicast packets should be delivered to the sending host
- Not yet supported
```

Description

Specify a copy of multicast packets should be delivered to the sending host - Not yet supported

IP_MULTICAST_TTL Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_TTL 33  //Set the Time to Live option for multicast packets - Not yet supported
```

Description

Set the Time to Live option for multicast packets - Not yet supported

IP_OPTIONS Macro

File

berkeley_api.h

C

```
#define IP_OPTIONS 4   //IP Header Options - Not yet supported
```

Description

IP Header Options - Not yet supported

IP_TOS Macro

File

berkeley_api.h

C

```
#define IP_TOS 1   //Type of Service - Not yet supported
```

Description

Type of Service - Not yet supported

IP_TTL Macro

File

berkeley_api.h

C

```
#define IP_TTL 2   //Time to Live - Not yet supported
```

Description

Time to Live - Not yet supported

IPPROTO_ICMPV6 Macro

File

berkeley_api.h

C

```
#define IPPROTO_ICMPV6 58  // Indicates IPv6 ICMP Protocol Options
```

Description

Indicates IPv6 ICMP Protocol Options

IPPROTO_IPV6 Macro

File

berkeley_api.h

C

```
#define IPPROTO_IPV6 41  // Indicates IP v6 Header options
```

Description

Indicates IP v6 Header options

IPV6_CHECKSUM Macro

File

berkeley_api.h

C

```
#define IPV6_CHECKSUM 7   //Sets the IPv6 Checksum options - Not yet supported
```

Description

Sets the IPv6 Checksum options - Not yet supported

IPV6_JOIN_GROUP Macro

File

berkeley_api.h

C

```
#define IPV6_JOIN_GROUP 20  //Join an IPv6 multicast group - Not yet supported
```

Description

Join an IPv6 multicast group - Not yet supported

IPV6_LEAVE_GROUP Macro

File

berkeley_api.h

C

```
#define IPV6_LEAVE_GROUP 21  //Leave an IPv6 multicast group - Not yet supported
```

Description

Leave an IPv6 multicast group - Not yet supported

IPV6_MULTICAST_HOPS Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_HOPS 18  //Set the hop limit for multicast IPv6 packets - Not yet supported
```

Description

Set the hop limit for multicast IPv6 packets - Not yet supported

IPV6_MULTICAST_IF Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_IF 17  //Set the Interface that multicast IPv6 packets should be sent on  - Not yet
supported
```

Description

Set the Interface that multicast IPv6 packets should be sent on - Not yet supported

IPV6_MULTICAST_LOOP Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_LOOP 19  //Specify a copy of multicast IPv6 packets should be delivered to the
sending host - Not yet supported
```

Description

Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported

IPV6_UNICAST_HOPS Macro

File

berkeley_api.h

C

```
#define IPV6_UNICAST_HOPS 16  //Set the hop limit for unicast IPv6 packets - Not yet supported
```

Description

Set the hop limit for unicast IPv6 packets - Not yet supported

IPV6_V6ONLY Macro

File

berkeley_api.h

C

```
#define IPV6_V6ONLY 26  //Sets the socket to IPv6 only - Not yet supported
```

Description

Sets the socket to IPv6 only - Not yet supported

SO_BROADCAST Macro

File

berkeley_api.h

C

```
#define SO_BROADCAST 6   //Enables the Socket for sending broadcast data
```

Description

Enables the Socket for sending broadcast data

SO_DEBUG Macro

File

berkeley_api.h

C

```
#define SO_DEBUG 1   //Indicates if low level debut is active - Not yet supported
```

Description

Indicates if low level debut is active - Not yet supported

SO_DONTROUTE Macro

File

berkeley_api.h

C

```
#define SO_DONTROUTE 5   //Bypass normal routing - Not yet supported
```

Description

Bypass normal routing - Not yet supported

SO_KEEPALIVE Macro

File

berkeley_api.h

C

```
#define SO_KEEPALIVE 9   //Keep the connection alive by sending periodic transmissions - Not yet supported
```

Description

Keep the connection alive by sending periodic transmissions - Not yet supported

SO_LINGER Macro

File

berkeley_api.h

C

```
#define SO_LINGER 13  //Indicates if the system should send any buffered data when a socket is closed
```

Description

Indicates if the system should send any buffered data when a socket is closed

SO_OOBINLINE Macro

File

berkeley_api.h

C

```
#define SO_OOBINLINE 10  //Indicates whether or not Out of Band Data should be received inline with normal
data - Not yet supported
```

Description

Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported

SO_RCVBUF Macro

File

berkeley_api.h

C

```
#define SO_RCVBUF 8   //Receive Buffer Size (TCP only)
```

Description

Receive Buffer Size (TCP only)

SO_RCVLOWAT Macro

File

berkeley_api.h

C

```
#define SO_RCVLOWAT 18  //Receive Low Water mark - Not yet supported
```

Description

Receive Low Water mark - Not yet supported

SO_RCVTIMEO Macro

File

berkeley_api.h

C

```
#define SO_RCVTIMEO 20  //Set the Receive Timeout - Not yet supported
```

Description

Set the Receive Timeout - Not yet supported

SO_REUSEADDR Macro

File

berkeley_api.h

C

```
#define SO_REUSEADDR 2   //Indicates if the local socket can be reused immediately after close - Not yet
supported
```

Description

Indicates if the local socket can be reused immediately after close - Not yet supported

SO_SNDBUF Macro

File

berkeley_api.h

C

```
#define SO_SNDBUF 7   //Send Buffer Size
```

Description

Send Buffer Size

SO_SNDLOWAT Macro

File

berkeley_api.h

C

```
#define SO_SNDLOWAT 19  //Send Low Water mark - Not yet supported
```

Description

Send Low Water mark - Not yet supported

SO_SNDTIMEO Macro

File

berkeley_api.h

C

```
#define SO_SNDTIMEO 21  //Set the Send Timeout - Not yet supported
```

Description

Set the Send Timeout - Not yet supported

SOL_SOCKET Macro

File

berkeley_api.h

C

```
#define SOL_SOCKET 1   // Indicates socket level options
```

Description

Indicates socket level options

TCP_NODELAY Macro

File

berkeley_api.h

C

```
#define TCP_NODELAY 1 //Indicates if TCP is to buffer packets - Not yet supported
```

Description

Indicates if TCP is to buffer packets - Not yet supported

linger Structure

File

berkeley_api.h

C

```
struct linger {
  int l_onoff;
  int l_linger;
};
```

Members

Members	Description
int l_onoff;	Determines if the option is set
int l_linger;	Time to wait before data is discarded

Description

This is record linger.

hostent Structure

File

berkeley_api.h

C

```
struct hostent {
  char * h_name;
  char ** h_alias;
  int h_addrtype;
  int h_length;
  char ** h_addr_list;
};
```

Members

Members	Description
char * h_name;	Points to a string containing the name of the host
char ** h_alias;	points to a null terminated list of pointers that point to the aliases of the host
int h_addrtype;	Contains the address type for the host, currently only AF_INET is supported
int h_length;	Contains the length of the h_addr_list
char ** h_addr_list;	Points to a NULL terminated list of pointers that point to the address of the host

Description

This is record hostent.

HOST_NOT_FOUND Macro

File

berkeley_api.h

C

```
#define HOST_NOT_FOUND 1    // Authoritative Answer host not found
```

Description

Authoritative Answer host not found

NO_DATA Macro

File

berkeley_api.h

C

```
#define NO_DATA 4    // Valid name, no data record of requested type
```

Description

Valid name, no data record of requested type

NO_RECOVERY Macro

File

berkeley_api.h

C

```
#define NO_RECOVERY 3    //Non recoverable errors,
```

Description

Non recoverable errors,

TRY_AGAIN Macro

File

berkeley_api.h

C

```
#define TRY_AGAIN 2    //Non authoritative host not found or server fail
```

Description

Non authoritative host not found or server fail

in6_addr Structure

File

berkeley_api.h

C

```
struct in6_addr {
  union {
    uint8_t u6_addr8[16];
    uint16_t u6_addr16[8];
    uint32_t u6_addr32[4];
  } in6_u;
};
```

Members

Members	Description
uint8_t u6_addr8[16];	IP address in Bytes
uint16_t u6_addr16[8];	IP address in 16 bit Words
uint32_t u6_addr32[4];	IP address in 32 bit Words

Description

This is record in6_addr.

sockaddr_in6 Structure

File

berkeley_api.h

C

```
struct sockaddr_in6 {
  short sin6_family;
  uint16_t sin6_port;
  uint32_t sin6_flowinfo;
  struct in6_addr sin6_addr;
  uint32_t sin6_scope_id;
};
```

Members

Members	Description
short sin6_family;	Address family; must be AF_INET.
uint16_t sin6_port;	Internet Protocol (IP) port.
uint32_t sin6_flowinfo;	IPv6 flow information
struct in6_addr sin6_addr;	IPv6 address in network byte order.
uint32_t sin6_scope_id;	IPv6 Scope Id

Description

In the Internet address family

sockaddr_storage Structure

File

berkeley_api.h

C

```
struct sockaddr_storage {
  short ss_family;
  __ss_aligntype __ss_align;
  char __ss_padding[_SS_PADSIZE];
};
```

Members

Members	Description
short ss_family;	Address family
__ss_aligntype __ss_align;	Force desired alignment.

Description

This is record sockaddr_storage.

SOCKADDR_IN6 Type

File

berkeley_api.h

C

```
typedef struct sockaddr_in6 SOCKADDR_IN6;
```

Description

This is type SOCKADDR_IN6.

AF_INET6 Macro

File

berkeley_api.h

C

```
#define AF_INET6 10          // Internet Address Family - IPv6
```

Description

Internet Address Family - IPv6

__ss_aligntype Macro

File

berkeley_api.h

C

```
#define __ss_aligntype __uint32_t
```

Description

Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.

_BERKELEY_API_HEADER_FILE Macro

File

berkeley_api.h

C

```
#define _BERKELEY_API_HEADER_FILE
```

Description

This is macro _BERKELEY_API_HEADER_FILE.

_SS_PADSIZE Macro

File

berkeley_api.h

C

```
#define _SS_PADSIZE (_SS_SIZE - (2 * sizeof (__ss_aligntype)))
```

Description

This is macro _SS_PADSIZE.

_SS_SIZE Macro

File

berkeley_api.h

C

```
#define _SS_SIZE 128
```

Description

This is macro _SS_SIZE.

BERKELEY_MODULE_CONFIG Structure

File

berkeley_api.h

C

```
typedef struct {
  uint8_t maxSockets;
} BERKELEY_MODULE_CONFIG;
```

Members

Members	Description
uint8_t maxSockets;	Maximum number of sockets supported

Description

Berkeley API module configuration structure

Files

Files

Name	Description
berkeley_api.h	The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs.
berkeley_api_config.h	Berkeley Sockets API Client and Server

Description

This section lists the source and header files used by the library.

berkeley_api.h

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs.

Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket.

	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket.
	connect	This function connects to the peer communications end point.
	freeaddrinfo	Frees the memory allocated by getaddrinfo
	getaddrinfo	Does an address look up for the provided node name.
	gethostbyname	The gethostbyname function returns a structure of type hostent for the given host name.
	gethostname	Returns the standard host name for the system.
	getsockname	Returns the current address to which the socket is bound.
	getsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	listen	The listen function sets the specified socket in a listen mode.
	recv	The recv() function is used to receive incoming data that has been queued for a socket.
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket.
	send	The send function is used to send outgoing data on an already connected socket.
	sendto	This function used to send the data for both connection oriented and connection-less sockets.
	setsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	socket	This function creates a new Berkeley socket.
	TCPIP_BSD_Socket	Returns the native socket number associated with the BSD socket.

Macros

	Name	Description
	__ss_aligntype	Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.
	_BERKELEY_API_HEADER_FILE	This is macro _BERKELEY_API_HEADER_FILE.
	_SS_PADSIZE	This is macro _SS_PADSIZE.
	_SS_SIZE	This is macro _SS_SIZE.
	AF_INET	Internet Address Family - IPv4, UDP, TCP, etc.
	AF_INET6	Internet Address Family - IPv6
	EAI_AGAIN	Temporary failure in name resolution.
	EAI_BADFLAGS	Invalid value for `ai_flags' field.
	EAI_FAIL	Non-recoverable failure in name res.
	EAI_FAMILY	ai_family' not supported.
	EAI_MEMORY	Memory allocation failure.
	EAI_NONAME	NAME or SERVICE is unknown.
	EAI_OVERFLOW	Argument buffer overflow.
	EAI_SERVICE	SERVICE not supported for `ai_socktype'.
	EAI_SOCKTYPE	ai_socktype' not supported.
	EAI_SYSTEM	System error returned in `errno'.
	HOST_NOT_FOUND	Authoritative Answer host not found
	ICMP6_FILTER	SEt the IPv6 Filtering options - Not yet supported
	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADD_MEMBERSHIP	Join a multicast group - Not yet supported
	IP_ADDR_ANY	IP Address for server binding
	IP_DROP_MEMBERSHIP	Leave a multicast group - Not yet supported
	IP_MULTICAST_IF	Set the Interface that multicast packets should be sent on - Not yet supported
	IP_MULTICAST_LOOP	Specify a copy of multicast packets should be delivered to the sending host - Not yet supported
	IP_MULTICAST_TTL	Set the Time to Live option for multicast packets - Not yet supported
	IP_OPTIONS	IP Header Options - Not yet supported
	IP_TOS	Type of Service - Not yet supported
	IP_TTL	Time to Live - Not yet supported
	IPPROTO_ICMPV6	Indicates IPv6 ICMP Protocol Options
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_IPV6	Indicates IP v6 Header options
	IPPROTO_TCP	Indicates TCP level options

	IPPROTO_UDP	Indicates UDP level options
	IPV6_CHECKSUM	Sets the IPv6 Checksum options - Not yet supported
	IPV6_JOIN_GROUP	Join an IPv6 multicast group - Not yet supported
	IPV6_LEAVE_GROUP	Leave an IPv6 multicast group - Not yet supported
	IPV6_MULTICAST_HOPS	Set the hop limit for multicast IPv6 packets - Not yet supported
	IPV6_MULTICAST_IF	Set the Interface that multicast IPv6 packets should be sent on - Not yet supported
	IPV6_MULTICAST_LOOP	Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported
	IPV6_UNICAST_HOPS	Set the hop limit for unicast IPv6 packets - Not yet supported
	IPV6_V6ONLY	Sets the socket to IPv6 only - Not yet supported
	NO_DATA	Valid name, no data record of requested type
	NO_RECOVERY	Non recoverable errors,
	SO_BROADCAST	Enables the Socket for sending broadcast data
	SO_DEBUG	Indicates if low level debut is active - Not yet supported
	SO_DONTROUTE	Bypass normal routing - Not yet supported
	SO_KEEPALIVE	Keep the connection alive by sending periodic transmissions - Not yet supported
	SO_LINGER	Indicates if the system should send any buffered data when a socket is closed
	SO_OOBINLINE	Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported
	SO_RCVBUF	Receive Buffer Size (TCP only)
	SO_RCVLOWAT	Receive Low Water mark - Not yet supported
	SO_RCVTIMEO	Set the Receive Timeout - Not yet supported
	SO_REUSEADDR	Indicates if the local socket can be reused immediately after close - Not yet supported
	SO_SNDBUF	Send Buffer Size
	SO_SNDLOWAT	Send Low Water mark - Not yet supported
	SO_SNDTIMEO	Set the Send Timeout - Not yet supported
	SOCK_DGRAM	Connectionless datagram socket. Use UDP for the Internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the Internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error
	SOL_SOCKET	Indicates socket level options
	TCP_NODELAY	Indicates if TCP is to buffer packets - Not yet supported
	TRY_AGAIN	Non authoritative host not found or server fail

Structures

	Name	Description
	addrinfo	This is record addrinfo.
	hostent	This is record hostent.
	in_addr	in_addr structure
	in6_addr	This is record in6_addr.
	linger	This is record linger.
	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family
	sockaddr_in6	In the Internet address family
	sockaddr_storage	This is record sockaddr_storage.
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure

Types

	Name	Description
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKADDR_IN6	This is type SOCKADDR_IN6.
	SOCKET	Socket descriptor

Description

Berkeley Socket Distribution API Header File

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

File Name

berkeley_api.h

Company

Microchip Technology Inc.

berkeley_api_config.h

Berkeley Sockets API Client and Server

Macros

	Name	Description
	MAX_BSD_SOCKETS	Berkeley API max number of sockets simultaneous supported

Description

Berkeley Sockets API Configuration file

This file contains the Berkeley API module configuration options

File Name

berkeley_api_config.h

Company

Microchip Technology Inc.

DHCP Module

This section describes the TCP/IP Stack Library DHCP module.

Introduction

TCP/IP Stack Library Dynamic Host Configuration Protocol (DHCP) Module for Microchip Microcontrollers

This library provides the API of the DHCP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCP client module will allow your application to dynamically obtain an IP address and a subnet mask from a DHCP server on the same network. Additionally, the DHCP client will get other parameters, such as gateway and DNS addresses.

Using the Library

This topic describes the basic architecture of the DHCP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcp.h`

The interface to the DHCP TCP/IP Stack library is defined in the `dhcp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

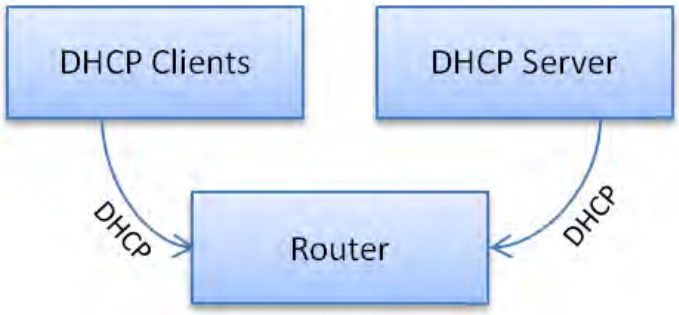
This library provides the API of the DHCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the DHCP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

DHCP Software Abstraction Block Diagram

Typical DCHP Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCP module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP
Status Functions	Routines for Obtaining DHCP Status Information
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

The Dynamic Host Configuration protocol (DHCP) is a standard networking protocol used to automatically allocate IP addresses for hosts in a network.

The DHCP server has a pool of IP addresses, which are leased for clients requesting them. The leases have a time-out after which the hosts need to renew the lease or acquire a new one.

The DHCP client module in the TCP/IP stack takes care of the communication with the DHCP server and renewing the lease when the time-out expires.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCP_BOOT_FILE_NAME_SIZE	size of the storage for the Boot file name should always be <= 128 default value is 128
	TCPIP_DHCP_STORE_BOOT_FILE_NAME	enable the usage of the Boot file name received from the DHCP server

Description

The configuration of the DHCP TCP/IP Stack is based on the file `system_config.h` (which may include a `dhcp_config.h` file).

This header file contains the configuration parameters for the DHCP TCP/IP Stack. Based on the selections made, the DHCP TCP/IP Stack will adjust its behavior.

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.

TCPIP_DHCP_BOOT_FILE_NAME_SIZE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_BOOT_FILE_NAME_SIZE 128
```

Description

size of the storage for the Boot file name should always be <= 128 default value is 128

TCPIP_DHCP_STORE_BOOT_FILE_NAME Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_STORE_BOOT_FILE_NAME
```

Description

enable the usage of the Boot file name received from the DHCP server

Building the Library

This section lists the files that are available in the DHCP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/dhcp.c`	DHCP 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 DHCP module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	Deregisters a previously registered DHCP handler.
	TCPIP_DHCP_HandlerRegister	Registers a DHCP Handler.
	TCPIP_DHCP_HostNameCallbackRegister	Registers a DHCP host name callback with the DHCP client.
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.

b) Status Functions

	Name	Description
	TCPIP_DHCP_IsBound	Determines if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determines if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determines if the DHCP client on the specified interface has been able to contact a DHCP server.
	TCPIP_DHCP_IsActive	Determines if the DHCP client is currently active on the specified interface.

c) Data Types and Constants

	Name	Description
	TCPIP_DHCP_EVENT_HANDLER	DHCP event handler prototype.
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type
	TCPIP_DHCP_HANDLE	DHCP handle.
	TCPIP_DHCP_MODULE_CONFIG	DHCP Module Configuration run-time parameters.
	TCPIP_DHCP_INFO	Reports DHCP module information.
	TCPIP_DHCP_STATUS	Lists the current status of the DHCP module.
	TCPIP_DHCP_HOST_NAME_CALLBACK	DHCP Host name callback function.
	TCPIP_DHCP_HOST_NAME_SIZE	Maximum size of a host name to be advertised to the DHCP server

Description

This section describes the Application Programming Interface (API) functions of the DHCP module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCP_Disable Function

Disables the DHCP Client for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DHCP client for the specified interface. If it is already disabled, no action is taken.

Remarks

When the DHCP client is disabled and the interface continues using its old configuration, it is possible that the lease may expire and the DHCP server provide the IP address to another client. The application should not keep the old lease unless it is sure that there is no danger of conflict.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
pNetIf	Interface to disable the DHCP client on.

Function

bool TCPIP_DHCP_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_Enable Function

Enables the DHCP client for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function enables the DHCP client for the specified interface, if it is disabled. If it is already enabled, no action is taken.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DHCP client on.

Function

void TCPIP_DHCP_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_HandlerDeRegister Function

Deregisters a previously registered DHCP handler.

File

dhcp.h

C

```
bool TCPIP_DHCP_HandlerDeRegister(TCPIP_DHCP_HANDLE hDhcp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the DHCP event handler.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hDhcp	A handle returned by a previous call to TCPIP_DHCP_HandlerRegister.

Function

TCPIP_DHCP_HandlerDeRegister(TCPIP_DHCP_HANDLE hDhcp)

TCPIP_DHCP_HandlerRegister Function

Registers a DHCP Handler.

File

dhcp.h

C

```
TCPIP_DHCP_HANDLE TCPIP_DHCP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_HANDLER handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

This function registers a DHCP event handler. The DHCP module will call the registered handler when a DHCP event (TCPIP_DHCP_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client and will be used by the DHCP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a DHCP event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the DHCP module.

Function

TCPIP_DHCP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_HANDLER handler, const void* hParam)

TCPIP_DHCP_HostNameCallbackRegister Function

Registers a DHCP host name callback with the DHCP client.

File

dhcp.h

C

```
bool TCPIP_DHCP_HostNameCallbackRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_HOST_NAME_CALLBACK nameCallback,
bool writeBack);
```

Returns

- true - if the call succeeds
- false - if error (no such interface, etc.)

Description

This function registers a DHCP host name callback function. The DHCP module will call the registered callback when a host name needs to be presented to the DHCP server.

Remarks

The nameCallback function has to be valid for the length of DHCP client communication with the server, i.e. when the reported status is bound or some error.

There is no corresponding deregister function. Simply call TCPIP_DHCP_HostNameCallbackRegister with the nameCallback set to 0.

If such a callback is not registered, then the NetBios name will be used for the DHCP host name.

The callback function has to return a character string that's compatible with the rules imposed for the host names:

- Host names may contain only alphanumeric characters, minus signs ("-"), and periods (".")
- They must begin with an alphabetic character and end with an alphanumeric character

To enforce these rules the extra processing is performed internally (on either user supplied host name or NetBios name):

- spaces, if present, will be stripped from the string
- illegal characters will be replaced by TCPIP_DHCP_HOST_REPLACE_CHAR character (default is lower case 'x')

If after processing the resulting string is null, the DHCP host name option is skipped.

The callback function has to return a character string that's at least 2 characters in size;

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
nameCallback	Callback to be called to get a DHCP host name
writeBack	if true, the resulting name will be written to the address returned by the nameCallback (it shouldn't be const!)

Function

bool TCPIP_DHCP_HostNameCallbackRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_HOST_NAME_CALLBACK nameCallback, bool writeBack)

TCPIP_DHCP_Renew Function

Renews the DHCP lease for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Renew(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function attempts to contact the server and renew the DHCP lease for the specified interface. The interface should have the DHCP enabled and in bound state for this call to succeed.

Preconditions

The DHCP module must be initialized and enabled, and have a valid lease.

Parameters

Parameters	Description
hNet	Interface on which to renew the DHCP lease.

Function

void TCPIP_DHCP_Renew(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_Request Function

Requests the supplied IPv4 address from a DHCP server.

File

dhcp.h

C

```
bool TCPIP_DHCP_Request(TCPIP_NET_HANDLE hNet, IPV4_ADDR reqAddress);
```

Returns

- true - if successful
- false - if the supplied IP address is invalid or the DHCP client is in the middle of a transaction

Description

If the DHCP client is not enabled on that interface, this call will first try to enable it. If this succeeds or the DHCP client was already enabled, the following steps are taken: The DHCP client probes the DHCP server and requests the supplied IPv4 address as a valid lease for the specified interface. If the server acknowledges the request, then this is the new IPv4 address of the interface. If the DHCP server rejects the request, then the whole DHCP process is resumed starting with the DHCP Discovery phase.

Remarks

The requested IPv4 address should be a previous lease that was granted to the host. This call should be used when the host is restarting.

Preconditions

The DHCP module must be initialized and enabled, and have a valid lease.

Parameters

Parameters	Description
hNet	Interface to renew the DHCP lease on.

Function

void TCPIP_DHCP_Request(TCPIP_NET_HANDLE hNet, IPV4_ADDR reqAddress)

b) Status Functions

TCPIP_DHCP_IsBound Function

Determines if the DHCP client has an IP address lease on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsBound(TCPIP_NET_HANDLE hNet);
```

Returns

- true - DHCP client has obtained an IP address lease (and likely other parameters) and these values are currently being used
- false - No IP address is currently leased

Description

This functin returns the status of the current IP address lease on the specified interface.

Preconditions

None.

Parameters

Parameters	Description
hNet	Interface to query

Function

bool TCPIP_DHCP_IsBound(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsEnabled Function

Determines if the DHCP client is enabled on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DHCP client service is enabled on the specified interface
- false - if the DHCP client service is not enabled on the specified interface

Description

This function returns the current state of the DHCP client on the specified interface.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsServerDetected Function

Determines if the DHCP client on the specified interface has been able to contact a DHCP server.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsServerDetected(TCPIP_NET_HANDLE hNet);
```

Returns

- true - At least one DHCP server is attached to the specified network interface
- false - No DHCP servers are currently detected on the specified network interface

Description

This function determines if the DHCP client on the specified interface received any reply from a DHCP server.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsServerDetected(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsActive Function

Determines if the DHCP client is currently active on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsActive(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DHCP client service is currently active on the specified interface
- false - if the DHCP client service is not active on the specified interface

Description

This function returns the current state of the DHCP client on the specified interface.

Remarks

The DHCP client service could be enabled bot not active. For example when there was no DHCP server detected on the network.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsActive(TCPIP_NET_HANDLE hNet)

c) Data Types and Constants

TCPIP_DHCP_EVENT_HANDLER Type

DHCP event handler prototype.

File

dhcp.h

C

```
typedef void (* TCPIP_DHCP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_TYPE evType, const void*
param);
```

Description

Type: TCPIP_DHCP_EVENT_HANDLER

Prototype of a DHCP event handler. Clients can register a handler with the DHCP service. Once an DHCP event occurs the DHCP service will called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

TCPIP_DHCP_EVENT_TYPE Enumeration

DHCP Event Type

File

dhcp.h

C

```
typedef enum {
  DHCP_EVENT_NONE = 0,
  DHCP_EVENT_DISCOVER,
  DHCP_EVENT_REQUEST,
  DHCP_EVENT_ACK,
  DHCP_EVENT_ACK_INVALID,
  DHCP_EVENT_DECLINE,
  DHCP_EVENT_NACK,
  DHCP_EVENT_TIMEOUT,
  DHCP_EVENT_BOUND,
  DHCP_EVENT_REQUEST_RENEW,
  DHCP_EVENT_REQUEST_REBIND,
  DHCP_EVENT_CONN_LOST,
  DHCP_EVENT_CONN_ESTABLISHED,
  DHCP_EVENT_SERVICE_DISABLED
} TCPIP_DHCP_EVENT_TYPE;
```

Members

Members	Description
DHCP_EVENT_NONE = 0	DHCP no event
DHCP_EVENT_DISCOVER	DHCP discovery sent: cycle started
DHCP_EVENT_REQUEST	DHCP request sent
DHCP_EVENT_ACK	DHCP request acknowledge was received
DHCP_EVENT_ACK_INVALID	DHCP acknowledge received but discarded as invalid
DHCP_EVENT_DECLINE	DHCP lease declined
DHCP_EVENT_NACK	DHCP negative acknowledge was received
DHCP_EVENT_TIMEOUT	DHCP server timeout
DHCP_EVENT_BOUND	DHCP lease obtained
DHCP_EVENT_REQUEST_RENEW	lease request renew sent
DHCP_EVENT_REQUEST_REBIND	lease request rebind sent
DHCP_EVENT_CONN_LOST	connection to the DHCP server lost
DHCP_EVENT_CONN_ESTABLISHED	connection re-established
DHCP_EVENT_SERVICE_DISABLED	DHCP service disabled, reverted to the default IP address

Description

Enumeration: TCPIP_DHCP_EVENT_TYPE

None.

TCPIP_DHCP_HANDLE Type

DHCP handle.

File

dhcp.h

C

```
typedef const void* TCPIP_DHCP_HANDLE;
```

Description

Type: TCPIP_DHCP_HANDLE

A handle that a client can use after the event handler has been registered.

TCPIP_DHCP_MODULE_CONFIG Structure

DHCP Module Configuration run-time parameters.

File

dhcp.h

C

```
typedef struct {
  bool dhcpEnable;
  int dhcpTmo;
  int dhcpCliPort;
  int dhcpSrvPort;
} TCPIP_DHCP_MODULE_CONFIG;
```

Members

Members	Description
bool dhcpEnable;	DHCP client enable at module start-up
int dhcpTmo;	timeout to wait for DHCP lease, seconds
int dhcpCliPort;	client port for DHCP client transactions
int dhcpSrvPort;	remote server port for DHCP server messages

Description

DHCP Module Configuration

This structure contains the data that's passed to the DHCP module at the TCP/IP stack initialization.

TCPIP_DHCP_INFO Structure

Reports DHCP module information.

File

dhcp.h

C

```
typedef struct {
  TCPIP_DHCP_STATUS status;
  uint32_t dhcpTime;
  uint32_t leaseStartTime;
  uint32_t leaseDuration;
  uint32_t renewTime;
  uint32_t rebindTime;
  IPV4_ADDR dhcpAddress;
  IPV4_ADDR serverAddress;
  const char* bootFileName;
} TCPIP_DHCP_INFO;
```

Members

Members	Description
TCPIP_DHCP_STATUS status;	current status
uint32_t dhcpTime;	current DHCP time, seconds the following fields are significant only if a lease has been obtained and is currently valid i.e. status >= TCPIP_DHCP_BOUND
uint32_t leaseStartTime;	time when lease was requested

uint32_t leaseDuration;	lease duration as returned by the server, seconds
uint32_t renewTime;	the time for moving to renew state, seconds
uint32_t rebindTime;	the time for moving to rebind state, seconds
IPV4_ADDR dhcpAddress;	IPv4 address obtained by DHCP
IPV4_ADDR serverAddress;	IPv4 address of the server that granted the lease
const char* bootFileName;	pointer to the bootfile name that was returned by the server This will be 0 if TCPIP_DHCP_STORE_BOOT_FILE_NAME option is not enabled!

Description

Structure: TCPIP_DHCP_INFO

This data structure is used for reporting current info and status of the DHCP module. Used in getting info about the DHCP module.

TCPIP_DHCP_STATUS Enumeration

Lists the current status of the DHCP module.

File

dhcp.h

C

```
typedef enum {
  TCPIP_DHCP_IDLE = 0,
  TCPIP_DHCP_WAIT_LINK,
  TCPIP_DHCP_GET_SOCKET,
  TCPIP_DHCP_SEND_DISCOVERY,
  TCPIP_DHCP_GET_OFFER,
  TCPIP_DHCP_SEND_REQUEST,
  TCPIP_DHCP_GET_REQUEST_ACK,
  TCPIP_DHCP_WAIT_LEASE_CHECK,
  TCPIP_DHCP_WAIT_LEASE_RETRY,
  TCPIP_DHCP_BOUND,
  TCPIP_DHCP_BOUND_RENEW,
  TCPIP_DHCP_SEND_RENEW,
  TCPIP_DHCP_GET_RENEW_ACK,
  TCPIP_DHCP_SEND_REBIND,
  TCPIP_DHCP_GET_REBIND_ACK
} TCPIP_DHCP_STATUS;
```

Members

Members	Description
TCPIP_DHCP_IDLE = 0	idle/inactive state
TCPIP_DHCP_WAIT_LINK	waiting for an active connection
TCPIP_DHCP_GET_SOCKET	trying to obtain a socket
TCPIP_DHCP_SEND_DISCOVERY	sending a Discover message
TCPIP_DHCP_GET_OFFER	waiting for a DHCP Offer
TCPIP_DHCP_SEND_REQUEST	sending a REQUEST message (REQUESTING)
TCPIP_DHCP_GET_REQUEST_ACK	waiting for a Request ACK message
TCPIP_DHCP_WAIT_LEASE_CHECK	waiting for received lease verification
TCPIP_DHCP_WAIT_LEASE_RETRY	waiting for another attempt after the lease verification failed
TCPIP_DHCP_BOUND	bound
TCPIP_DHCP_BOUND_RENEW	a lease renew is initiated
TCPIP_DHCP_SEND_RENEW	sending a REQUEST message (RENEW state)
TCPIP_DHCP_GET_RENEW_ACK	waiting for ACK in RENEW state
TCPIP_DHCP_SEND_REBIND	sending REQUEST message (REBIND state)
TCPIP_DHCP_GET_REBIND_ACK	waiting for ACK in REBIND state

Description

Enumeration: TCPIP_DHCP_STATUS

This enumeration lists the current status of the DHCP module. Used in getting information about the DHCP state machine.

TCPIP_DHCP_HOST_NAME_CALLBACK Type

DHCP Host name callback function.

File

dhcp.h

C

```
typedef char* (* TCPIP_DHCP_HOST_NAME_CALLBACK)(TCPIP_NET_HANDLE hNet);
```

Description

Type: TCPIP_DHCP_HOST_NAME_CALLBACK

Prototype of a DHCP callback function that returns the host name to be presented to the server by the DHCP client.

This callback will be called by the DHCP client when communicating to the server and a host name is needed.

Remarks

There are certain restictions that apply to the host name strings. See the TCPIP_DHCP_HostNameCallbackRegister function.

TCPIP_DHCP_HOST_NAME_SIZE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_HOST_NAME_SIZE 20
```

Description

Maximum size of a host name to be advertised to the DHCP server

Files

Files

Name	Description
dhcp.h	Dynamic Host Configuration Protocol (DHCP) client API definitions.
dhcp_config.h	DCHP configuration file

Description

This section lists the source and header files used by the library.

dhcp.h

Dynamic Host Configuration Protocol (DHCP) client API definitions.

Enumerations

	Name	Description
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type
	TCPIP_DHCP_STATUS	Lists the current status of the DHCP module.

Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	Deregisters a previously registered DHCP handler.
	TCPIP_DHCP_HandlerRegister	Registers a DHCP Handler.
	TCPIP_DHCP_HostNameCallbackRegister	Registers a DHCP host name callback with the DHCP client.
	TCPIP_DHCP_InfoGet	Returns information about the DHCP client on the specified interface.
	TCPIP_DHCP_IsActive	Determines if the DHCP client is currently active on the specified interface.

	TCPIP_DHCP_IsBound	Determines if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determines if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determines if the DHCP client on the specified interface has been able to contact a DHCP server.
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.
	TCPIP_DHCP_RequestTimeoutSet	Sets the DHCP client request and base time-out values.
	TCPIP_DHCP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCP_INFO	Reports DHCP module information.
	TCPIP_DHCP_MODULE_CONFIG	DHCP Module Configuration run-time parameters.

Types

	Name	Description
	TCPIP_DHCP_EVENT_HANDLER	DHCP event handler prototype.
	TCPIP_DHCP_HANDLE	DHCP handle.
	TCPIP_DHCP_HOST_NAME_CALLBACK	DHCP Host name callback function.

Description

Dynamic Host Configuration Protocol (DHCP) Client API Header File

Provides automatic IP address, subnet mask, gateway address, DNS server address, and other configuration parameters on DHCP enabled networks.

File Name

dhcp.h

Company

Microchip Technology Inc.

dhcp_config.h

DCHP configuration file

Macros

	Name	Description
	TCPIP_DHCP_BOOT_FILE_NAME_SIZE	size of the storage for the Boot file name should always be <= 128 default value is 128
	TCPIP_DHCP_CLIENT_ENABLED	Default value for the enable/disable the DHCP client at stack start-up.
	TCPIP_DHCP_HOST_NAME_SIZE	Maximum size of a host name to be advertised to the DHCP server
	TCPIP_DHCP_STORE_BOOT_FILE_NAME	enable the usage of the Boot file name received from the DHCP server
	TCPIP_DHCP_TASK_TICK_RATE	The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCP_TIMEOUT	Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds

Description

Dynamic Host Configuration Protocol (DCHP) Configuration file

This file contains the DCHP module configuration options

File Name

dhcp_config.h

Company

Microchip Technology Inc.

DHCP Server Module

This section describes the TCP/IP Stack Library DHCP Server module.

Introduction

TCP/IP Stack Library DHCP Server Module for Microchip Microcontrollers

This library provides the API of the DHCP Server module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCP Server module is used to assign the IP address to the DHCP client from the configured IP address database. When the server receives a request from a client, the DHCP server determines the network to which the DHCP client is connected, and then allocates an IP address that is appropriate for the client, and sends configuration information appropriate for that client. DHCP servers typically grant IP addresses to clients only for a limited interval. DHCP clients are responsible for renewing their IP address before that interval has expired, and must stop using the address once the interval has expired, if they have not been able to renew it.

Using the Library

This topic describes the basic architecture of the DHCP Server TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcps.h`

The interface to the DHCP Server TCP/IP Stack library is defined in the `dhcps.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCP Server TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

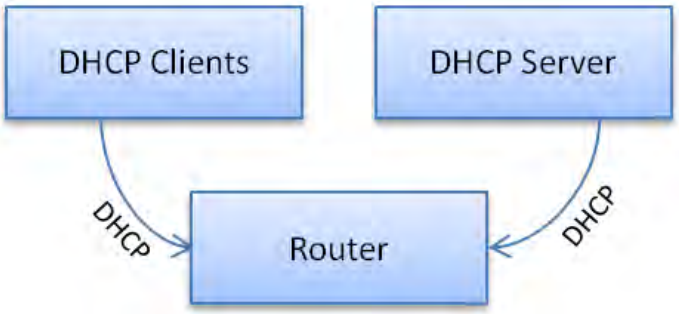
This library provides the API of the DHCP Server TCP/IP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DHCP Server Software Abstraction Block Diagram

This module provides software abstraction of the DHCP Server module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

Typical DCHP Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCP Server module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP server
Status Functions	Routines for Obtaining DHCP Status Information

Data Types and Constants	This section provides various definitions describing this API

How the Library Works

To use DHCP Server, include the files `dhcps.c`, `dhcp.c`, and `dhcp.h` in your project, and add or uncomment the definition "#define TCPIP_STACK_USE_DHCP_SERVER" to `tcpip_config.h`.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START	These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.
	TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS	DHCP server Address per interface. DHCP server Address selection should be in the same subnet.
	TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS	DHCP server subnet Address per interface.
	TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS	DHCP server DNS primary Address
	TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS	DHCP server DNS Secondary Address
	TCPIP_DHCPS_LEASE_DURATION	Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again
	TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT	The Maximum Number of entries in the lease table Default total number of entries for all the the interface
	TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK	Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER
	TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again
	TCPIP_DHCPS_TASK_PROCESS_RATE	DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the DHCP Server TCP/IP Stack is based on the file `dhcps_config.h`.

This header file contains the configuration selection for the DHCP Server TCP/IP Stack. Based on the selections made, the DHCP Server TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DHCP Server TCP/IP Stack.

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.

TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START "192.168.1.100"
```

Description

These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.

TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS "192.168.1.1"
```

Description

DHCP server Address per interface. DHCP server Address selection should be in the same subnet.

TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS "255.255.255.0"
```

Description

DHCP server subnet Address per interface.

TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS "192.168.1.1"
```

Description

DHCP server DNS primary Address

TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS "192.168.1.1"
```

Description

DHCP server DNS Secondary Address

TCPIP_DHCPS_LEASE_DURATION Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_DURATION TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO
```

Description

Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again

TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT 15
```

Description

The Maximum Number of entries in the lease table Default total number of entries for all the the interface

TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK (5)
```

Description

Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER

TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again

TCPIP_DHCPS_TASK_PROCESS_RATE Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_TASK_PROCESS_RATE (200)
```

Description

DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the DHCP Server module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/dhcps.c`	DHCP Server 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 DHCP Server module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCPS_Disable	Disables the DHCP Server for the specified interface.
	TCPIP_DHCPS_Enable	Enables the DHCP Server for the specified interface.
	TCPIP_DHCPS_RemovePoolEntries	Removes all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_LeaseEntryRemove	Remove one entry from the DHCP server leased entry.
	TCPIP_DHCP_RequestTimeoutSet	Sets the DHCP client request and base time-out values.
	TCPIP_DHCP_Task	Standard TCP/IP stack module task function.
	TCPIP_DHCPS_Task	Standard TCP/IP stack module task function.

b) Status Functions

	Name	Description
	TCPIP_DHCPS_IsEnabled	Determines if the DHCP Server is enabled on the specified interface.
	TCPIP_DHCPS_GetPoolEntries	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_LeaseEntryGet	Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.
	TCPIP_DHCP_InfoGet	Returns information about the DHCP client on the specified interface.

c) Data Types and Constants

	Name	Description
	TCPIP_DHCPS_LEASE_ENTRY	DHCP Server module runtime and initialization configuration data.
	TCPIP_DHCPS_LEASE_HANDLE	DHCP Server Lease Handle
	TCPIP_DHCPS_POOL_ENTRY_TYPE	DHCP server pool types are used to get and remove the leased IP address details.
	TCPIP_DHCPS_ADDRESS_CONFIG	DHCP server configuration and IP address range.
	TCPIP_DHCP_CLIENT_ENABLED	Default value for the enable/disable the DHCP client at stack start-up.

	TCPIP_DHCP_TASK_TICK_RATE	The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCP_TIMEOUT	Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds
	TCPIP_DHCPS_MODULE_CONFIG	DHCP Server module runtime and initialization configuration data.

Description

This section describes the Application Programming Interface (API) functions of the DHCP Server module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCPS_Disable Function

Disables the DHCP Server for the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function disables the DHCP Server for the specified interface. If it is already disabled, no action is taken.

Remarks

When the interface continues using its old configuration, it is possible that the lease may take sometime to expire. And The communication will be there until it is not expired.Lease time is configured in dhcps_config.h.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to disable the DHCP Server

Function

bool TCPIP_DHCPS_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_Enable Function

Enables the DHCP Server for the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function enables the DHCP Server for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to enable the DHCP Server

Function

void TCPIP_DHCPS_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_RemovePoolEntries Function

Removes all the entries or only used entries of a certain type belonging to a network interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_RemovePoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to remove the DHCP server entries from the pool as per TCPIP_DHCPS_POOL_ENTRY_TYPE.

Remarks

None.

Preconditions

the DHCP Server module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
type	type of entries to remove: DHCP_SERVER_POOL_ENTRY_ALL, DHCP_SERVER_POOL_ENTRY_IN_USE

Function

bool TCPIP_DHCPS_RemovePoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);

TCPIP_DHCPS_LeaseEntryRemove Function

Remove one entry from the DHCP server leased entry.

File

dhcps.h

C

```
bool TCPIP_DHCPS_LeaseEntryRemove(TCPIP_NET_HANDLE netH, TCPIP_MAC_ADDR* hwAdd);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to remove one entry from the leased HASH table with respect to the interface and the MAC address.

Remarks

This function is called from the command line to remove one entry and from the Wi-FI Driver module to remove a node that is disconnected from the AP.

Preconditions

The DHCP Server module should have been initialized.

Parameters

Parameters	Description
netH	Interface handle to use
hwAdd	MAC address that need to be removed from the HASH table

Function

bool TCPIP_DHCPS_LeaseEntryRemove(TCPIP_NET_HANDLE netH, TCPIP_MAC_ADDR* hwAdd)

TCPIP_DHCP_RequestTimeoutSet Function

Sets the DHCP client request and base time-out values.

File

dhcp.h

C

```
bool TCPIP_DHCP_RequestTimeoutSet(TCPIP_NET_HANDLE hNet, uint16_t initTmo, uint16_t dhcpBaseTmo);
```

Returns

- true - if successful
- false - if a wrong interface handle or time-out value was provided

Description

This function allows the run time adjustment of the DHCP time-out values. It specifies for how long the client has to wait for a valid DHCP server reply during the initialization process until acquisition of the host address is considered to have failed. It also sets the DHCP base timeout for DHCP transactions with the server. This is automatically incremented by the DHCP client using an exponential back-off algorithm. Recommended value is 2 <= dhcpBaseTmo <= 64 seconds.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle
initTmo	Initialization time-out to wait for a valid lease acquisition, in seconds
dhcpBaseTmo	DHCP time-out value for DHCP requests, in seconds

Function

bool TCPIP_DHCP_RequestTimeoutSet(TCPIP_NET_HANDLE hNet, uint16_t initTmo,

uint16_t dhcpBaseTmo)

TCPIP_DHCP_Task Function

Standard TCP/IP stack module task function.

File

dhcp.h

C

```
void TCPIP_DHCP_Task();
```

Returns

None.

Description

This function performs DHCP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DHCP module should have been initialized

Function

void TCPIP_DHCP_Task(void)

TCPIP_DHCPS_Task Function

Standard TCP/IP stack module task function.

File

dhcps.h

C

```
void TCPIP_DHCPS_Task();
```

Returns

None.

Description

This function performs DHCP Server module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DHCP Server module should have been initialized.

Function

void TCPIP_DHCPS_Task(void)

b) Status Functions

TCPIP_DHCPS_IsEnabled Function

Determines if the DHCP Server is enabled on the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function returns the current state of the DHCP Server on the specified interface.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query

Function

bool TCPIP_DHCPS_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_GetPoolEntries Function

Get all the entries or only used entries of a certain type belonging to a network interface.

File

dhcps.h

C

```
int TCPIP_DHCPS_GetPoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to get the DHCP server entries from the pool as per TCPIP_DHCPS_POOL_ENTRY_TYPE.

Remarks

None.

Preconditions

The DHCP server module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
type	type of entries to remove: • DHCP_SERVER_POOL_ENTRY_ALL, • DHCP_SERVER_POOL_ENTRY_IN_USE

Function

int TCPIP_DHCPS_GetPoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);

TCPIP_DHCPS_LeaseEntryGet Function

Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.

File

dhcps.h

C

```
TCPIP_DHCPS_LEASE_HANDLE TCPIP_DHCPS_LeaseEntryGet(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_LEASE_ENTRY*
pLeaseEntry, TCPIP_DHCPS_LEASE_HANDLE leaseHandle);
```

Returns

- non-zero TCPIP_DHCPS_LEASE_HANDLE - To be used in the subsequent calls
- 0 - If end of list or wrong interface, or DHCP server is not running on that interface

Description

This function returns a lease entry for the TCPIP_DHCPS_LEASE_HANDLE. if the lease entry is not present for that

TCPIP_DHCPS_LEASE_HANDLE, then it will return the next valid lease entry.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
netH	Lease entry for this Interface
pLeaseEntry	Client lease entry details
leaseHandle	Lease index

Function

TCPIP_DHCPS_LEASE_HANDLE TCPIP_DHCPS_LeaseEntryGet(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_LEASE_ENTRY* pLeaseEntry, TCPIP_DHCPS_LEASE_HANDLE leaseHandle);

TCPIP_DHCP_InfoGet Function

Returns information about the DHCP client on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_InfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_INFO* pDhcpInfo);
```

Returns

- true - if the interface is enabled and exists and the DHCP client service information filled in the supplied storage
- false - if the specified interface is not enabled, does not exist, or does not have a DHCP client

Description

This function returns the current state and lease info of the DHCP client on the specified interface.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.
pDhcpInfo	storage to return the DHCP info

Function

bool TCPIP_DHCP_InfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_INFO* pDhcpInfo);

c) Data Types and Constants

TCPIP_DHCPS_LEASE_ENTRY Structure

DHCP Server module runtime and initialization configuration data.

File

dhcps.h

C

```
typedef struct {
  TCPIP_MAC_ADDR hwAdd;
  IPV4_ADDR ipAddress;
  uint32_t leaseTime;
} TCPIP_DHCPS_LEASE_ENTRY;
```

Members

Members	Description
TCPIP_MAC_ADDR hwAdd;	Client MAC address
IPV4_ADDR ipAddress;	Leased IP address
uint32_t leaseTime;	Lease period

Description

Structure: TCPIP_DHCPS_MODULE_CONFIG

DHCP server configuration and initialization data. Configuration is part of tcpip_stack_init.c.

TCPIP_DHCPS_LEASE_HANDLE Type

DHCP Server Lease Handle

File

dhcps.h

C

```
typedef const void* TCPIP_DHCPS_LEASE_HANDLE;
```

Description

Type: TCPIP_DHCPS_LEASE_HANDLE

A handle that server is using to provide the index of a lease entry.

Remarks

This handle is used by command handler to get the Index of Lease entry.

TCPIP_DHCPS_POOL_ENTRY_TYPE Enumeration

DHCP server pool types are used to get and remove the leased IP address details.

File

dhcps.h

C

```
typedef enum {
  DHCP_SERVER_POOL_ENTRY_ALL,
  DHCP_SERVER_POOL_ENTRY_IN_USE
} TCPIP_DHCPS_POOL_ENTRY_TYPE;
```

Members

Members	Description
DHCP_SERVER_POOL_ENTRY_ALL	Get or Remove all the Leased address
DHCP_SERVER_POOL_ENTRY_IN_USE	Get or remove only Leased IP address

Description

Enumeration: TCPIP_DHCPS_POOL_ENTRY_TYPE

DHCP_SERVER_POOL_ENTRY_ALL - Get or Remove all the leased address which includes both solved and unsolved entries.
DHCP_SERVER_POOL_ENTRY_IN_USE - Get or Remove only solved leased IP address.

TCPIP_DHCPS_ADDRESS_CONFIG Structure

DHCP server configuration and IP address range.

File

dhcps.h

C

```
typedef struct {
  int interfaceIndex;
```

```
    char* serverIPAddress;
    char* startIPAddRange;
    char* ipMaskAddress;
    char* priDNS;
    char* secondDNS;
    bool poolEnabled;
  } TCPIP_DHCPS_ADDRESS_CONFIG;
```

Members

Members	Description
int interfaceIndex;	Interface Index
char* serverIPAddress;	Server IP address
char* startIPAddRange;	Start IP address
char* ipMaskAddress;	Netmask
char* priDNS;	Primary DNS server Address
char* secondDNS;	Secondary DNS server Address
bool poolEnabled;	true if pool is valid , false if pool is invalid

Description

Structure: TCPIP_DHCPS_ADDRESS_CONFIG

DHCP server configuration and network initialization data. Configuration is part of tcpip_stack_init.c.

TCPIP_DHCP_CLIENT_ENABLED Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_CLIENT_ENABLED 1
```

Description

Default value for the enable/disable the DHCP client at stack start-up.

Remarks

the interface initialization setting in TCPIP_NETWORK_CONFIG takes precedence!

TCPIP_DHCP_TASK_TICK_RATE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_TASK_TICK_RATE (200)
```

Description

The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DHCP_TIMEOUT Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_TIMEOUT (10)
```

Description

Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds

TCPIP_DHCPS_MODULE_CONFIG Structure

DHCP Server module runtime and initialization configuration data.

File

dhcps.h

C

```
typedef struct {
  bool enabled;
  bool deleteOldLease;
  size_t leaseEntries;
  uint32_t entrySolvedTmo;
  TCPIP_DHCPS_ADDRESS_CONFIG * dhcpServer;
} TCPIP_DHCPS_MODULE_CONFIG;
```

Members

Members	Description
bool enabled;	enable DHCP server
bool deleteOldLease;	delete old cache if still in place, specific DHCP parameters
size_t leaseEntries;	max number of lease entries
uint32_t entrySolvedTmo;	solved entry removed after this tmo in seconds
TCPIP_DHCPS_ADDRESS_CONFIG * dhcpServer;	DHCP server lease address configuration details uint32_t dhcpServerCnt; // Max DHCP server support

Description

Structure: TCPIP_DHCPS_MODULE_CONFIG

DHCP server configuration and initialization data . Configuration is part of tcpip_stack_init.c.

Files

Files

Name	Description
dhcps.h	Dynamic Host Configuration Protocol(DHCP) Server APIs.
dhcps_config.h	DHCPS configuration file

Description

This section lists the source and header files used by the library.

dhcps.h

Dynamic Host Configuration Protocol(DHCP) Server APIs.

Enumerations

	Name	Description
	TCPIP_DHCPS_POOL_ENTRY_TYPE	DHCP server pool types are used to get and remove the leased IP address details.

Functions

	Name	Description
	TCPIP_DHCPS_Disable	Disables the DHCP Server for the specified interface.
	TCPIP_DHCPS_Enable	Enables the DHCP Server for the specified interface.
	TCPIP_DHCPS_GetPoolEntries	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_IsEnabled	Determines if the DHCP Server is enabled on the specified interface.
	TCPIP_DHCPS_LeaseEntryGet	Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.
	TCPIP_DHCPS_LeaseEntryRemove	Remove one entry from the DHCP server leased entry.
	TCPIP_DHCPS_RemovePoolEntries	Removes all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCPS_ADDRESS_CONFIG	DHCP server configuration and IP address range.
	TCPIP_DHCPS_LEASE_ENTRY	DHCP Server module runtime and initialization configuration data.
	TCPIP_DHCPS_MODULE_CONFIG	DHCP Server module runtime and initialization configuration data.

Types

	Name	Description
	TCPIP_DHCPS_LEASE_HANDLE	DHCP Server Lease Handle

Description

Dynamic Host Configuration Protocol (DHCP) Server API Header File

The DHCP server assigns a free IP address to a requesting client from the range defined in the dhcps_config.h file. Lease time per IP address is decided as per the TMO configuration which is defined in dhcps_config.h.

File Name

dhcps.h

Company

Microchip Technology Inc.

dhcps_config.h

DHCPS configuration file

Macros

	Name	Description
	TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START	These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.
	TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS	DHCP server Address per interface. DHCP server Address selection should be in the same subnet.
	TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS	DHCP server subnet Address per interface.
	TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS	DHCP server DNS primary Address
	TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS	DHCP server DNS Secondary Address
	TCPIP_DHCPS_LEASE_DURATION	Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again
	TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT	The Maximum Number of entries in the lease table Default total number of entries for all the the interface
	TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK	Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER
	TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again
	TCPIP_DHCPS_TASK_PROCESS_RATE	DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Dynamic Host Configuration Protocol (DHCPS) Configuration file

This file contains the DHCPS module configuration options

File Name

dhcps_config.h

Company

Microchip Technology Inc.

DHCPv6 Module

This section describes the TCP/IP Stack Library DHCPv6 module.

Introduction

TCP/IP Stack Library IPv6 Dynamic Host Configuration Protocol (DHCPv6) Module for Microchip Microcontrollers

This library provides the API of the DHCPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCPv6 client module will allow your application to dynamically obtain an IPv6 address from a DHCPv6 server. Additionally, the DHCPv6 client will get other parameters, such as DNS servers and the domain search list.

Using the Library

This topic describes the basic architecture of the DHCPv6 TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcpv6.h`

The interface to the DHCPv6 TCP/IP Stack library is defined in the `dhcpv6.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCPv6 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

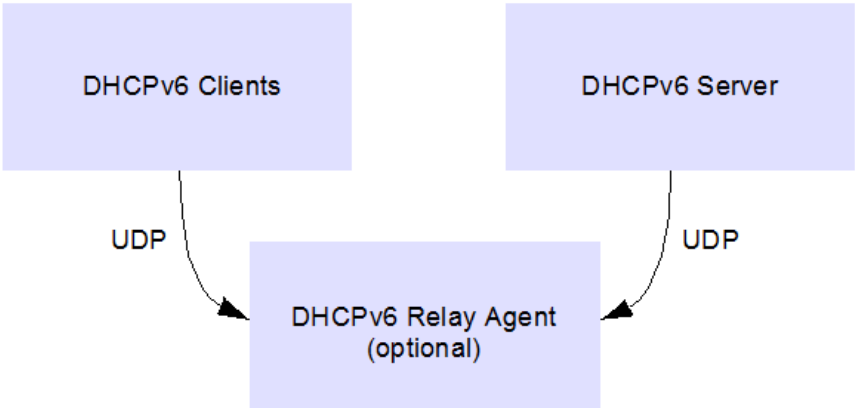
This library provides the API of the DHCPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the DHCPv6 module existent in the TCP/IP Stack implementation. The DHCPv6 module works together with the IPv6 Stateless Address Auto-configuration protocol and provides stateful temporary/non-temporary (IATA, IANA) global addresses for the IPv6 host.

DHCPv6 Software Abstraction Block Diagram

Typical DCHPv6 Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCPv6 module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCPv6
Status Functions	Routines for Obtaining DHCPv6 Status Information
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

The IPv6 Dynamic Host Configuration Protocol (DHCPv6) is a standard networking protocol used to automatically allocate IPv6 addresses for hosts in a network.

The DHCPv6 server has a pool of IP addresses, which are leased for clients requesting them. The leases have a limited lifetime after which the hosts need to renew the lease or acquire a new one.

The DHCPv6 client module in the TCP/IP stack takes care of the communication with the DHCPv6 server and renewing the lease when the lifetime expires.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCPV6_CLIENT_DUID_TYPE	Default DUID type to be used by the client
	TCPIP_DHCPV6_CLIENT_PORT	Clients listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_DNS_SERVERS_NO	number of DNS servers to store from a DHCP server reply
	TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE	space for the Domain Search List option - multiple of 16
	TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE	preference value that forces the server selection 8 bit value!
	TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO	maximum number of free IA descriptors per client
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME	default lifetimes for the solicited addresses
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME	This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.
	TCPIP_DHCPV6_IANA_DEFAULT_T1	default values for IANA T1, T2
	TCPIP_DHCPV6_IANA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.
	TCPIP_DHCPV6_IANA_DESCRIPTORS_NO	maximum number of IANA descriptors per client
	TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO	default addresses for IANA in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_IANA_SOLICIT_T1	default values for IANA T1, T2 in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_T2	This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.
	TCPIP_DHCPV6_IATA_DEFAULT_T1	default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)
	TCPIP_DHCPV6_IATA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.
	TCPIP_DHCPV6_IATA_DESCRIPTORS_NO	maximum number of IATA descriptors per client
	TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO	default addresses for IATA in a solicit message
	TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE	default value of the buffer to assemble messages, etc.
	TCPIP_DHCPV6_MESSAGE_BUFFERS	default number of buffers
	TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE	minimum size of a UDP buffer
	TCPIP_DHCPV6_SERVER_PORT	Servers and relay agents listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_SKIP_DAD_PROCESS	defining this symbol will skip the DAD processing for DHCPv6 generated addresses
	TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN	number of character to reserve for a server status code associated message

	TCPIP_DHCPV6_TASK_TICK_RATE	The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_STACK_USE_DHCPV6_CLIENT	This symbol enables the DHCPv6 client as being part of the build
	TCPIP_DHCPV6_USER_NOTIFICATION	allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used

Description

The configuration of the DHCPv6 TCP/IP Stack is based on the file `system_config.h` (which may include a `dhcpv6_config.h` file).

This header file contains the configuration parameters for the DHCPv6 TCP/IP Stack. Based on the selections made, the DHCPv6 TCP/IP Stack will adjust its behavior.

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.

TCPIP_DHCPV6_CLIENT_DUID_TYPE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_CLIENT_DUID_TYPE TCPIP_DHCPV6_DUID_TYPE_LL
```

Description

Default DUID type to be used by the client

TCPIP_DHCPV6_CLIENT_PORT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_CLIENT_PORT 546
```

Description

Clients listen for DHCP messages on UDP port:

TCPIP_DHCPV6_DNS_SERVERS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_DNS_SERVERS_NO 2
```

Description

number of DNS servers to store from a DHCP server reply

TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE 64
```

Description

space for the Domain Search List option - multiple of 16

TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE 255
```

Description

preference value that forces the server selection 8 bit value!

TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO 2
```

Description

maximum number of free IA descriptors per client

TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME 0
```

Description

default lifetimes for the solicited addresses

TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME 0
```

Description

This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.

TCPIP_DHCPV6_IANA_DEFAULT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DEFAULT_T1 0
```

Description

default values for IANA T1, T2

TCPIP_DHCPV6_IANA_DEFAULT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DEFAULT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.

TCPIP_DHCPV6_IANA_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DESCRIPTORS_NO 4
```

Description

maximum number of IANA descriptors per client

TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO 0
```

Description

default addresses for IANA in a solicit message

TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS "::0"
```

Description

default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO

TCPIP_DHCPV6_IANA_SOLICIT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_T1 0
```

Description

default values for IANA T1, T2 in a solicit message

TCPIP_DHCPV6_IANA_SOLICIT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.

TCPIP_DHCPV6_IATA_DEFAULT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DEFAULT_T1 0
```

Description

default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)

TCPIP_DHCPV6_IATA_DEFAULT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DEFAULT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.

TCPIP_DHCPV6_IATA_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DESCRIPTORS_NO 2
```

Description

maximum number of IATA descriptors per client

TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO 0
```

Description

default addresses for IATA in a solicit message

TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS 0
```

Description

default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO

TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE 512
```

Description

default value of the buffer to assemble messages, etc.

TCPIP_DHCPV6_MESSAGE_BUFFERS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MESSAGE_BUFFERS 8
```

Description

default number of buffers

TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE 512
```

Description

minimum size of a UDP buffer

TCPIP_DHCPV6_SERVER_PORT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_SERVER_PORT 547
```

Description

Servers and relay agents listen for DHCP messages on UDP port:

TCPIP_DHCPV6_SKIP_DAD_PROCESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_SKIP_DAD_PROCESS
```

Description

defining this symbol will skip the DAD processing for DHCPv6 generated addresses

TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN 0
```

Description

number of character to reserve for a server status code associated message

TCPIP_DHCPV6_TASK_TICK_RATE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_TASK_TICK_RATE (100)
```

Description

The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_STACK_USE_DHCPV6_CLIENT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_STACK_USE_DHCPV6_CLIENT
```

Description

This symbol enables the DHCPv6 client as being part of the build

TCPIP_DHCPV6_USER_NOTIFICATION Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_USER_NOTIFICATION false
```

Description

allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used

Building the Library

This section lists the files that are available in the DHCPv6 module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/dhcpv6.c`	DHCPv6 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 DHCPv6 module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCPV6_HandlerDeRegister	DHCPV6 event deregistration
	TCPIP_DHCPV6_HandlerRegister	DHCPV6 event registration
	TCPIP_DHCPV6_Task	Standard TCP/IP stack module task function.

b) Status Functions

	Name	Description
	TCPIP_DHCPV6_ClientInfoGet	client status reporting
	TCPIP_DHCPV6_IaInfoGet	IA status reporting

c) Data Types and Constants

	Name	Description
	TCPIP_DHCPV6_CLIENT_INFO	DHCPv6 client info
	TCPIP_DHCPV6_CLIENT_STATE	DHCPv6 Current Status
	TCPIP_DHCPV6_CONFIG_FLAGS	DHCPV6 start up flags
	TCPIP_DHCPV6_DUID_TYPE	types of DUID for DHCPv6
	TCPIP_DHCPV6_EVENT_HANDLER	DHCPv6 event handler prototype.
	TCPIP_DHCPV6_HANDLE	a DHCPV6 handle

	TCPIP_DHCPV6_IA_EVENT	IA event info
	TCPIP_DHCPV6_IA_INFO	DHCPv6 IA info
	TCPIP_DHCPV6_IA_STATE	IA run states
	TCPIP_DHCPV6_IA_SUBSTATE	IA run substates most IA run states that must send a message go through these substates
	TCPIP_DHCPV6_IA_TYPE	supported types of IA
	TCPIP_DHCPV6_MODULE_CONFIG	DHCPv6 module configuration
	TCPIP_DHCPV6_SERVER_STATUS_CODE	DHCPV6 server status code

Description

This section describes the Application Programming Interface (API) functions of the DHCPv6 module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCPV6_HandlerDeRegister Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_HandlerDeRegister(TCPIP_DHCPV6_HANDLE hDhcp);
```

Description

DHCPV6 event deregistration

TCPIP_DHCPV6_HandlerRegister Function

File

dhcpv6.h

C

```
TCPIP_DHCPV6_HANDLE TCPIP_DHCPV6_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_EVENT_HANDLER handler,
const void* hParam);
```

Description

DHCPV6 event registration

TCPIP_DHCPV6_Task Function

Standard TCP/IP stack module task function.

File

dhcpv6.h

C

```
void TCPIP_DHCPV6_Task();
```

Returns

None.

Description

This function performs DHCPv6 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DHCPv6 module should have been initialized

Function

void TCPIP_DHCPV6_Task(void)

b) Status Functions

TCPIP_DHCPV6_ClientInfoGet Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_ClientInfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_CLIENT_INFO* pClientInfo);
```

Description

client status reporting

TCPIP_DHCPV6_IaInfoGet Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_IaInfoGet(TCPIP_NET_HANDLE hNet, int iaIx, TCPIP_DHCPV6_IA_INFO* pIaInfo);
```

Description

IA status reporting

c) Data Types and Constants

TCPIP_DHCPV6_CLIENT_INFO Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_CLIENT_STATE clientState;
  int nIanas;
  int nIatas;
  int nFreeIas;
  uint32_t dhcpTime;
  TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;
  void* statusBuff;
  size_t statusBuffSize;
  int nDnsServers;
  IPV6_ADDR* dnsBuff;
  size_t dnsBuffSize;
  int domainSearchListSize;
  void* domainBuff;
  size_t domainBuffSize;
} TCPIP_DHCPV6_CLIENT_INFO;
```

Members

Members	Description
TCPIP_DHCPV6_CLIENT_STATE clientState;	client state at the moment of the call

int nIanas;	number of IANA the client has
int nIatas;	number of IATA the client has
int nFreeIas;	number of free IAs the client has
uint32_t dhcpTime;	current DHCPV6 time; seconds
TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;	last status code for the client
void* statusBuff;	buffer to copy the latest status message associated with the client
size_t statusBuffSize;	size of this buffer
int nDnsServers;	number of DNS servers
IPV6_ADDR* dnsBuff;	buffer to copy the DNS Servers obtained from the DHCPV6 server
size_t dnsBuffSize;	size of this buffer
int domainSearchListSize;	size of domainSearchList
void* domainBuff;	buffer to store the domain Search list obtained from the DHCPv6 server
size_t domainBuffSize;	size of this buffer

Description

DHCPv6 client info

TCPIP_DHCPV6_CLIENT_STATE Enumeration

DHCPv6 Current Status

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_CLIENT_STATE_INIT = 0,
  TCPIP_DHCPV6_CLIENT_STATE_IDLE,
  TCPIP_DHCPV6_CLIENT_STATE_RUN,
  TCPIP_DHCPV6_CLIENT_STATE_WAIT_LINK,
  TCPIP_DHCPV6_CLIENT_STATE_NUMBER
} TCPIP_DHCPV6_CLIENT_STATE;
```

Members

Members	Description
TCPIP_DHCPV6_CLIENT_STATE_INIT = 0	initialization state/unknown
TCPIP_DHCPV6_CLIENT_STATE_IDLE	idle/inactive state
TCPIP_DHCPV6_CLIENT_STATE_RUN	up and running in one of the run states
TCPIP_DHCPV6_CLIENT_STATE_WAIT_LINK	up and running, waiting for a connection
TCPIP_DHCPV6_CLIENT_STATE_NUMBER	number of states

Description

Enumeration: TCPIP_DHCPV6_CLIENT_STATE

This enumeration lists the current status of the DHCPv6 module. Used in getting info about the DHCPv6 state machine.

TCPIP_DHCPV6_CONFIG_FLAGS Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_FLAG_NONE = 0,
  TCPIP_DHCPV6_FLAG_START_ENABLE = 0x01,
  TCPIP_DHCPV6_FLAG_DAD_DISABLE = 0x02,
  TCPIP_DHCPV6_FLAG_IA_IGNORE_RENEW_LTIME = 0x04,
  TCPIP_DHCPV6_FLAG_IA_IGNORE_REBIND_LTIME = 0x08,
  TCPIP_DHCPV6_FLAG_IA_NOTIFY_SUB_STATE = 0x80
} TCPIP_DHCPV6_CONFIG_FLAGS;
```

Members

Members	Description
TCPIP_DHCPV6_FLAG_START_ENABLE = 0x01	enable the DHCPv6 at stack start up
TCPIP_DHCPV6_FLAG_DAD_DISABLE = 0x02	disable the DAD processing for DHCP generated addresses Use only for testing or in special cases Default should be disabled
TCPIP_DHCPV6_FLAG_IA_IGNORE_RENEW_LTIME = 0x04	if enabled, the IA (and its associated address) renew process will be valid as dictated by t1/defaultIataT1 and its address preferred lifetime will be ignored If disabled, the IA and its address will attempt renew when the minimum of address preferred lifetime and t1/defaultIataT1 expired Default should be disabled
TCPIP_DHCPV6_FLAG_IA_IGNORE_REBIND_LTIME = 0x08	if enabled, the IA (and its associated address) rebind process will be valid as dictated by t2/defaultIataT2 and its address valid lifetime will be ignored If disabled, the IA and its address will attempt rebind when the minimum of address valid lifetime and t2/defaultIataT2 expired Default should be disabled
TCPIP_DHCPV6_FLAG_IA_NOTIFY_SUB_STATE = 0x80	if enabled, the IA notifications will be generated for IA substate changes too (finer grain) if disabled, notifications will be generated for IA state changes only Default should be disabled

Description

DHCPV6 start up flags

TCPIP_DHCPV6_DUID_TYPE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_DUID_TYPE_NONE = 0,
  TCPIP_DHCPV6_DUID_TYPE_LLT = 1,
  TCPIP_DHCPV6_DUID_TYPE_EN = 2,
  TCPIP_DHCPV6_DUID_TYPE_LL = 3
} TCPIP_DHCPV6_DUID_TYPE;
```

Members

Members	Description
TCPIP_DHCPV6_DUID_TYPE_NONE = 0	invalid
TCPIP_DHCPV6_DUID_TYPE_LLT = 1	LinkLayer + time
TCPIP_DHCPV6_DUID_TYPE_EN = 2	Enterprise Number
TCPIP_DHCPV6_DUID_TYPE_LL = 3	Link Layer Address

Description

types of DUID for DHCPv6

TCPIP_DHCPV6_EVENT_HANDLER Type

DHCPv6 event handler prototype.

File

dhcpv6.h

C

```
typedef void (* TCPIP_DHCPV6_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_CLIENT_STATE clientState,
const TCPIP_DHCPV6_IA_EVENT* pDhcpIaEv, const void* param);
```

Description

DHCPv6 reported event structure:

Type: TCPIP_DHCPV6_EVENT_HANDLER

Prototype of a DHCP event handler. Clients can register a handler with the DHCP service. Once an DHCP event occurs the DHCP service will called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

if pDhcpIaEvent == 0, no info is carried for a specific IA if pDhcpIaEvent != 0, the info carried by this pointer is not persistent and is valid only within

the context of this event handler!

TCPIP_DHCPV6_HANDLE Type

File

dhcpv6.h

C

```
typedef const void* TCPIP_DHCPV6_HANDLE;
```

Description

a DHCPV6 handle

TCPIP_DHCPV6_IA_EVENT Union

File

dhcpv6.h

C

```
typedef union {
  uint32_t eventVal;
  struct {
    uint8_t iaType;
    uint8_t iaState;
    uint8_t iaSubState;
    uint8_t iaIndex;
  }
} TCPIP_DHCPV6_IA_EVENT;
```

Members

Members	Description
uint8_t iaType;	a TCPIP_DHCPV6_IA_TYPE value
uint8_t iaState;	a TCPIP_DHCPV6_IA_STATE value
uint8_t iaSubState;	a TCPIP_DHCPV6_IA_SUBSTATE value
uint8_t iaIndex;	index/ID of this IA for this client

Description

IA event info

TCPIP_DHCPV6_IA_INFO Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_IA_TYPE iaType;
  TCPIP_DHCPV6_IA_STATE iaState;
  TCPIP_DHCPV6_IA_SUBSTATE iaSubState;
  int iaIndex;
  uint32_t iaId;
  uint32_t tAcquire;
  uint32_t t1;
  uint32_t t2;
  IPV6_ADDR ipv6Addr;
  uint32_t prefLTime;
  uint32_t validLTime;
  TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;
  void* statusBuff;
  size_t statusBuffSize;
} TCPIP_DHCPV6_IA_INFO;
```

Members

Members	Description
TCPIP_DHCPV6_IA_TYPE iaType;	IA type
TCPIP_DHCPV6_IA_STATE iaState;	IA state
TCPIP_DHCPV6_IA_SUBSTATE iaSubState;	IA substate
int iaIndex;	index of this IA for this client
uint32_t iaId;	ID of this IA the following fields are meaningful only for iaState >= TCPIP_DHCPV6_IA_STATE_BOUND
uint32_t tAcquire;	time of which the address was acquired
uint32_t t1;	IANA only: extend lifetime contact server time
uint32_t t2;	IANA only: extend lifetime contact any server time
IPV6_ADDR ipv6Addr;	16 bytes IPV6 address associated with this IA
uint32_t prefLTime;	preferred life time for the IPv6 address; seconds
uint32_t validLTime;	valid life time for the IPv6 address; seconds
TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;	last status code for this IA
void* statusBuff;	buffer to copy the latest status message associated with this IA
size_t statusBuffSize;	size of this buffer

Description

DHCPv6 IA info

TCPIP_DHCPV6_IA_STATE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_STATE_SOLICIT,
  TCPIP_DHCPV6_IA_STATE_REQUEST,
  TCPIP_DHCPV6_IA_STATE_DAD,
  TCPIP_DHCPV6_IA_STATE_DECLINE,
  TCPIP_DHCPV6_IA_STATE_BOUND,
  TCPIP_DHCPV6_IA_STATE_RENEW,
  TCPIP_DHCPV6_IA_STATE_REBIND,
  TCPIP_DHCPV6_IA_STATE_CONFIRM,
  TCPIP_DHCPV6_IA_STATE_RELEASE,
  TCPIP_DHCPV6_IA_STATE_ERROR_TRANSIENT,
  TCPIP_DHCPV6_IA_STATE_ERROR_FATAL,
  TCPIP_DHCPV6_IA_STATE_NUMBER
} TCPIP_DHCPV6_IA_STATE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_STATE_SOLICIT	solicitation
TCPIP_DHCPV6_IA_STATE_REQUEST	perform request
TCPIP_DHCPV6_IA_STATE_DAD	start the DAD state
TCPIP_DHCPV6_IA_STATE_DECLINE	decline the DAD addresses
TCPIP_DHCPV6_IA_STATE_BOUND	bound;
TCPIP_DHCPV6_IA_STATE_RENEW	renew address
TCPIP_DHCPV6_IA_STATE_REBIND	rebind address
TCPIP_DHCPV6_IA_STATE_CONFIRM	confirm address
TCPIP_DHCPV6_IA_STATE_RELEASE	release address
TCPIP_DHCPV6_IA_STATE_ERROR_TRANSIENT	an error occurred for which either user intervention is required
TCPIP_DHCPV6_IA_STATE_ERROR_FATAL	fatal error occurred; not properly configured options/buffers, etc;
TCPIP_DHCPV6_IA_STATE_NUMBER	number of run states

Description

IA run states

TCPIP_DHCPV6_IA_SUBSTATE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_SUBSTATE_START,
  TCPIP_DHCPV6_IA_SUBSTATE_IDELAY,
  TCPIP_DHCPV6_IA_SUBSTATE_TRANSMIT,
  TCPIP_DHCPV6_IA_SUBSTATE_WAIT_REPLY,
  TCPIP_DHCPV6_IA_SUBSTATE_NUMBER
} TCPIP_DHCPV6_IA_SUBSTATE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_SUBSTATE_START	message start/preparation
TCPIP_DHCPV6_IA_SUBSTATE_IDELAY	message wait for iDelay
TCPIP_DHCPV6_IA_SUBSTATE_TRANSMIT	send/transmit message
TCPIP_DHCPV6_IA_SUBSTATE_WAIT_REPLY	wait message reply
TCPIP_DHCPV6_IA_SUBSTATE_NUMBER	number of standard message sub-states

Description

IA run substates most IA run states that must send a message go through these substates

TCPIP_DHCPV6_IA_TYPE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_TYPE_NONE,
  TCPIP_DHCPV6_IA_TYPE_IANA,
  TCPIP_DHCPV6_IA_TYPE_IATA,
  TCPIP_DHCPV6_IA_TYPE_NUMBER
} TCPIP_DHCPV6_IA_TYPE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_TYPE_NONE	unused IA association
TCPIP_DHCPV6_IA_TYPE_IANA	IANA association
TCPIP_DHCPV6_IA_TYPE_IATA	IATA association
TCPIP_DHCPV6_IA_TYPE_NUMBER	number of types

Description

supported types of IA

TCPIP_DHCPV6_MODULE_CONFIG Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_CONFIG_FLAGS configFlags;
  uint16_t dhcpCliPort;
```

```
  uint16_t dhcpSrvPort;
  uint16_t duidType;
  uint16_t nIanaDcpts;
  uint16_t nIataDcpts;
  uint16_t nFreeDcpts;
  uint32_t defaultIanaT1;
  uint32_t defaultIanaT2;
  uint32_t defaultIataT1;
  uint32_t defaultIataT2;
  uint32_t ianaSolicitT1;
  uint32_t ianaSolicitT2;
  uint32_t solicitPrefLTime;
  uint32_t solicitValidLTime;
  int nMsgBuffers;
  int msgBufferSize;
} TCPIP_DHCPV6_MODULE_CONFIG;
```

Members

Members	Description
TCPIP_DHCPV6_CONFIG_FLAGS configFlags;	DHCPv6 client configuration flags
uint16_t dhcpCliPort;	client port for DHCPv6 client transactions
uint16_t dhcpSrvPort;	remote server port for DHCPv6 server messages
uint16_t duidType;	TCPIP_DHCPV6_DUID_TYPE: type to use for the DHCPv6 clients
uint16_t nIanaDcpts;	number of IANAs per client; default should be 1
uint16_t nIataDcpts;	number of IATAs per client; default should be 0
uint16_t nFreeDcpts;	number of free IAs per client - they could be added at run time; default should be 0
uint32_t defaultIanaT1;	The default time at which the client contacts the server to extend the lifetimes of the assigned IA_NA addresses If the IANA t1 value received from the server is 0, then this value will be used to override A value of 0 means the t1 is infinite default value should be 0
uint32_t defaultIanaT2;	The default time at which the client contacts any available server to extend the lifetimes of the assigned IA_NA addresses If the IANA t2 value received from the server is 0, then this value will be used to override if !0 it should be > defaultIanaT1! Has to be > t1 A value of 0 means the t2 is infinite default value should be 0
uint32_t defaultIataT1;	The default time at which the client contacts the server to extend the lifetimes of the assigned IATA addresses If 0, the timeout will be infinite (0xffffffff)
uint32_t defaultIataT2;	The default time at which the client contacts any available server to extend the lifetimes of the assigned IA_TA addresses if !0 it should be > defaultIataT1! If 0, the timeout will be infinite (0xffffffff)
uint32_t ianaSolicitT1;	The default T1 time to solicit from the server default should be 0
uint32_t ianaSolicitT2;	The default T2 time to solicit from the server default should be 0
uint32_t solicitPrefLTime;	default addresses preferred lifetime to solicit from the server default should be 0
uint32_t solicitValidLTime;	default addresses valid lifetime to solicit from the server default should be 0
int nMsgBuffers;	number of message buffers to allocate for this client these buffers are used for the TX/RX operations Enough buffers need to be allocated for gathering server advertisements and being able to transmit messages default should be 4
int msgBufferSize;	size of the message buffers default is 512 bytes

Description

DHCPv6 module configuration

TCPIP_DHCPV6_SERVER_STATUS_CODE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_SERVER_STAT_SUCCESS = 0,
  TCPIP_DHCPV6_SERVER_STAT_UNSPEC_FAIL = 1,
  TCPIP_DHCPV6_SERVER_STAT_NO_ADDRS_AVAIL = 2,
  TCPIP_DHCPV6_SERVER_STAT_NO_BINDING = 3,
  TCPIP_DHCPV6_SERVER_STAT_NOT_ON_LINK = 4,
  TCPIP_DHCPV6_SERVER_STAT_USE_MULTICAST = 5,
  TCPIP_DHCPV6_SERVER_STAT_MAX_CODE = 5,
  TCPIP_DHCPV6_SERVER_STAT_EXT_ERROR = -1
```

```
} TCPIP_DHCPV6_SERVER_STATUS_CODE;
```

Members

Members	Description
TCPIP_DHCPV6_SERVER_STAT_SUCCESS = 0	success
TCPIP_DHCPV6_SERVER_STAT_UNSPEC_FAIL = 1	Failure, reason unspecified; this status code is sent by either a client or a server to indicate a failure not explicitly specified in the RFC.
TCPIP_DHCPV6_SERVER_STAT_NO_ADDRS_AVAIL = 2	Server has no addresses available to assign to the IA(s).
TCPIP_DHCPV6_SERVER_STAT_NO_BINDING = 3	Client record (binding) unavailable.
TCPIP_DHCPV6_SERVER_STAT_NOT_ON_LINK = 4	The prefix for the address is not appropriate for the link to which the client is attached.
TCPIP_DHCPV6_SERVER_STAT_USE_MULTICAST = 5	Sent by a server to a client to force the client to send messages to the server using the All_DHCP_Relay_Agents_and_Servers address.
TCPIP_DHCPV6_SERVER_STAT_MAX_CODE = 5	maximum valid value
TCPIP_DHCPV6_SERVER_STAT_EXT_ERROR = -1	an error occurred, status code not found, etc.

Description

DHCPV6 server status code

Files

Files

Name	Description
dhcpv6.h	Dynamic Host Configuration Protocol (DHCPv6) client API definitions.
dhcpv6_config.h	DCHPv6 configuration file

Description

This section lists the source and header files used by the library.

dhcpv6.h

Dynamic Host Configuration Protocol (DHCPv6) client API definitions.

Enumerations

	Name	Description
	TCPIP_DHCPV6_CLIENT_STATE	DHCPv6 Current Status
	TCPIP_DHCPV6_CONFIG_FLAGS	DHCPV6 start up flags
	TCPIP_DHCPV6_DUID_TYPE	types of DUID for DHCPv6
	TCPIP_DHCPV6_IA_STATE	IA run states
	TCPIP_DHCPV6_IA_SUBSTATE	IA run substates most IA run states that must send a message go through these substates
	TCPIP_DHCPV6_IA_TYPE	supported types of IA
	TCPIP_DHCPV6_SERVER_STATUS_CODE	DHCPV6 server status code

Functions

	Name	Description
	TCPIP_DHCPV6_ClientInfoGet	client status reporting
	TCPIP_DHCPV6_HandlerDeRegister	DHCPV6 event deregistration
	TCPIP_DHCPV6_HandlerRegister	DHCPV6 event registration
	TCPIP_DHCPV6_IaInfoGet	IA status reporting
	TCPIP_DHCPV6_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCPV6_CLIENT_INFO	DHCPv6 client info
	TCPIP_DHCPV6_IA_INFO	DHCPv6 IA info
	TCPIP_DHCPV6_MODULE_CONFIG	DHCPv6 module configuration

Types

	Name	Description
	TCPIP_DHCPV6_EVENT_HANDLER	DHCPv6 event handler prototype.
	TCPIP_DHCPV6_HANDLE	a DHCPV6 handle

Unions

	Name	Description
	TCPIP_DHCPV6_IA_EVENT	IA event info

Description

IPv6 Dynamic Host Configuration Protocol (DHCPv6) Client API Header File

Provides automatic IP address, subnet mask, gateway address, DNS server address, and other configuration parameters on DHCPv6 enabled networks.

File Name

dhcpv6.h

Company

Microchip Technology Inc.

dhcpv6_config.h

DCHPv6 configuration file

Macros

	Name	Description
	TCPIP_DHCPV6_CLIENT_DUID_TYPE	Default DUID type to be used by the client
	TCPIP_DHCPV6_CLIENT_PORT	Clients listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_DNS_SERVERS_NO	number of DNS servers to store from a DHCP server reply
	TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE	space for the Domain Search List option - multiple of 16
	TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE	preference value that forces the server selection 8 bit value!
	TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO	maximum number of free IA descriptors per client
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME	default lifetimes for the solicited addresses
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME	This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.
	TCPIP_DHCPV6_IANA_DEFAULT_T1	default values for IANA T1, T2
	TCPIP_DHCPV6_IANA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.
	TCPIP_DHCPV6_IANA_DESCRIPTORS_NO	maximum number of IANA descriptors per client
	TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO	default addresses for IANA in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_IANA_SOLICIT_T1	default values for IANA T1, T2 in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_T2	This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.
	TCPIP_DHCPV6_IATA_DEFAULT_T1	default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)
	TCPIP_DHCPV6_IATA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.
	TCPIP_DHCPV6_IATA_DESCRIPTORS_NO	maximum number of IATA descriptors per client
	TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO	default addresses for IATA in a solicit message
	TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE	default value of the buffer to assemble messages, etc.
	TCPIP_DHCPV6_MESSAGE_BUFFERS	default number of buffers
	TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE	minimum size of a UDP buffer
	TCPIP_DHCPV6_SERVER_PORT	Servers and relay agents listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_SKIP_DAD_PROCESS	defining this symbol will skip the DAD processing for DHCPv6 generated addresses

	TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN	number of character to reserve for a server status code associated message
	TCPIP_DHCPV6_TASK_TICK_RATE	The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCPV6_USER_NOTIFICATION	allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used
	TCPIP_STACK_USE_DHCPV6_CLIENT	This symbol enables the DHCPv6 client as being part of the build

Description

IPv6 Dynamic Host Configuration Protocol (DCHPv6) Configuration file

This file contains the DCHPv6 module configuration options

File Name

dhcpv6_config.h

Company

Microchip Technology Inc.

DNS Module

This section describes the TCP/IP Stack Library DNS module.

Introduction

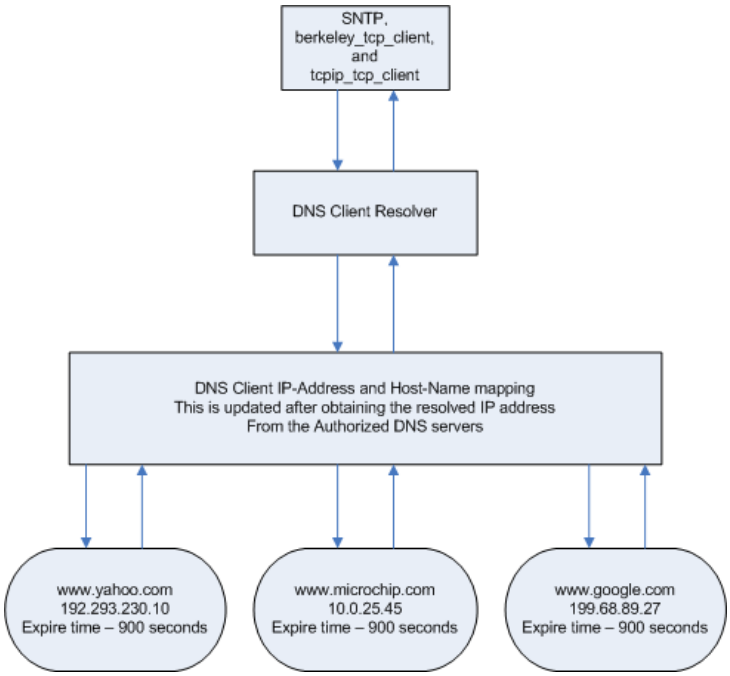
TCP/IP Stack Library Domain Name System (DNS) Module for Microchip Microcontrollers

This library provides the API of the DNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Domain Name Service associates host names (i.e., www.microchip.com) with IP addresses (i.e., 10.0.54.2). The DNS Client module provides DNS resolution capabilities to the stack. As illustrated in the figure, a typical resolution process is as follows:

- The DNS Client, also known as the DNS Resolver, will try to resolve the IP address from its local resources, which includes the Name to Address mappings in the local DNS cache. DNS clients are capable of caching the previous DNS query results and this module supports maximum cache entries as per the configuration parameter of DNS_CLIENT_CACHE_ENTRIES. Each cache entry is capable of storing the maximum number of IPv4 and IPv6 address, which is done per the configured parameters of DNS_CLIENT_CACHE_PER_IPV4_ADDRESS and DNS_CLIENT_CACHE_PER_IPV6_ADDRESS. The DNS resolver will use these available local resources and if it cannot find the IP Address as per DNS Record type, it will then query a DNS server that it knows for the IP Address.
- The DNS server will directly answer the query if it is the authoritative server for the particular domain (ex.: www.microchip.com); otherwise, it will check its local cache of the previous queries. If it cannot find the IP Address, it will then query the DNS servers on the Internet.
- Next, the DNS Server will query one of the root servers requesting a list of Authoritative servers for the .COM domain. The Root server will respond with the list of servers addresses hosting the .COM domain.
- The Authoritative DNS server will then respond with the IP Address or a list of IP Address (if more than one server hosts the website www.microchip.com) for www.microchip.com.
- The address that the DNS server returns back to the DNS client is then passed to the DNS application, which initially requested the IP Address information -- in this case, the TCP/IP client application.
- Finally, the Application then knows where to go to fetch the web site.



Using the Library

This topic describes the basic architecture of the DNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dns.h`

The interface to the DNS TCP/IP Stack library is defined in the `dns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DNS TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the DNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

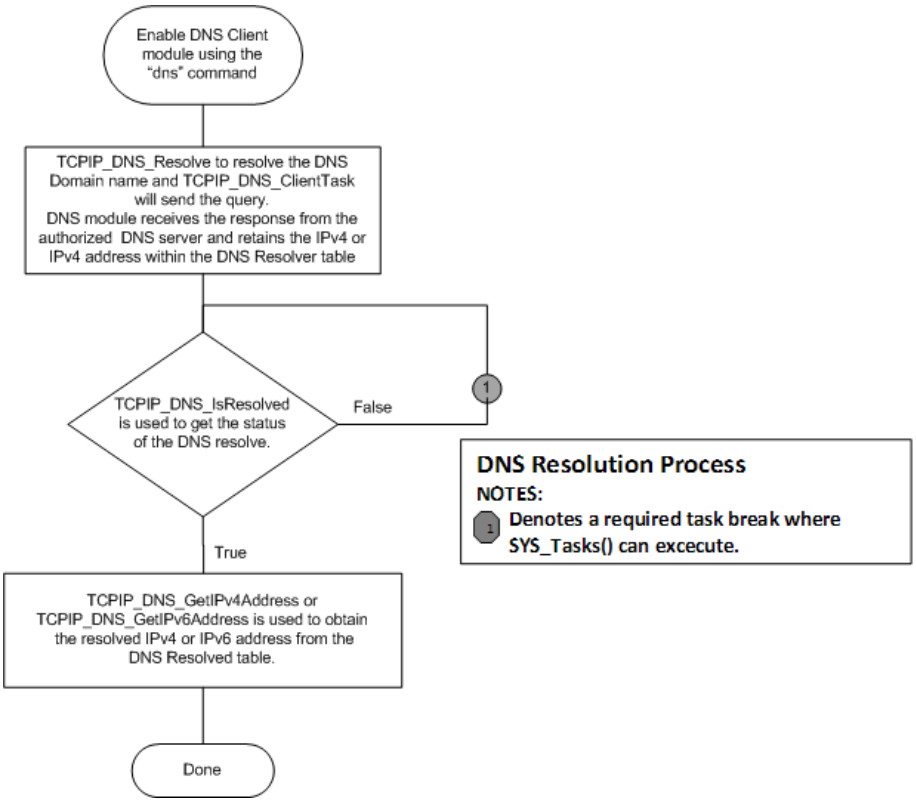
Description

DNS is part of the Application Layer. How the DNS Client works and how it is used is described in the following algorithm and figure.

1. By default, the DNS client is enabled and the DNS module can be enabled from the command prompt using the command "dns PIC32INT on". If the DNS Client module is enabled, a UDP socket will opened for port number 53 for the DNS client applications. The DNS client module can be disabled from the command prompt using "dns PIC32INT off".

Note: For the Wi-Fi module, the interface name "PIC32INT" will be changed to "MRF24W".

2. Applications or demonstrations such as berkeley_tcp_client, berkeley_udp_client, tcpip_tcp_client, and tcpip_udp_client, use the TCPIP_DNS_Resolve function with Host name and Record type as arguments to resolve the domain host name. The DNS module supports the Record type of "A" and "AAAA". Record Type "A" is used for IPv4 and "AAAA" is used for an IPv6 address.
3. The resolved domain name and the mapped IPv4 or IPv6 address will be stored in a table with the expire time, which is calculated from the query response.
4. The number of the IPv4 and IPv6 address will be cached as per the DNS_CLIENT_CACHE_PER_IPV4_ADDRESS function and per the DNS_CLIENT_CACHE_PER_IPV6_ADDRESS function from the query response.
5. The TCPIP_DNS_IsResolved function will return DNS_RES_OK if the domain name resolve is successful.
6. The TCPIP_DNS_GetNumberOfIPAddresses function is used to obtain the number of the IPv4 or IPv6 address, which is present in the DNS Resolver table. The DNS Resolver table is a mapping table of the IP address, domain name, and the expire time.
7. The functions, TCPIP_DNS_GetIPv4Address and TCPIP_DNS_GetIPv6Address, are used to obtain the IPv4 or Ipv6 address from the resolved table as per the domain host name and index number. The index number should be less than DNS_CLIENT_CACHE_PER_IPV4_ADDRESS or DNS_CLIENT_CACHE_PER_IPV6_ADDRESS. The index number is the location of the resolved table, which will help to fetch the IPv4 or IPv6 address from the exact location. For example, if index == 0, TCPIP_DNS_GetIPv4Address or TCPIP_DNS_GetIPv6Address will return 0th location IPv4 or Ipv6 address if there is more than one IPv4 or IPv6 address.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DNS module.

Library Interface Section	Description
General Functions	This section provides general interface routines to DNS
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DNS_CLIENT_CACHE_ENTRIES	Number of DNS resolver entries
	TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO	DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_SERVER_TMO	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds
	TCPIP_DNS_CLIENT_TASK_PROCESS_RATE	DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL	Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0
	TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO	Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO
	TCPIP_DNS_CLIENT_USER_NOTIFICATION	allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used
	TCPIP_DNS_CLIENT_ADDRESS_TYPE	This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development
	TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO	Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO
	TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN	Max DNS host name size
	TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES	Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1: • the default interface should always be considered for DNS resolution Depending on how many active interfaces select those to be considered for DNS resolution

Description

The configuration of the DNS TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DNS TCP/IP Stack. Based on the selections made, the DNS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DNS TCP/IP Stack.

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.

TCPIP_DNS_CLIENT_CACHE_ENTRIES Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_ENTRIES 5
```

Description

Number of DNS resolver entries

TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO 0
```

Description

DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.

TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS 5
```

Description

Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.

TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS 1
```

Description

Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.

TCPIP_DNS_CLIENT_SERVER_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_SERVER_TMO (1*60)
```

Description

When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds

TCPIP_DNS_CLIENT_TASK_PROCESS_RATE Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_TASK_PROCESS_RATE (200)
```

Description

DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL (20 * 60)
```

Description

Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0

TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO (10)
```

Description

Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO

TCPIP_DNS_CLIENT_USER_NOTIFICATION Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_USER_NOTIFICATION false
```

Description

allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used

TCPIP_DNS_CLIENT_ADDRESS_TYPE Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_ADDRESS_TYPE IP_ADDRESS_TYPE_IPV4
```

Description

This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development

TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO (3)
```

Description

Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO

TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN 32
```

Description

Max DNS host name size

TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES 4
```

Description

Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1:

- the default interface should always be considered for DNS resolution

Depending on how many active interfaces select those to be considered for DNS resolution

Building the Library

This section lists the files that are available in the DNS module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/dns.c	DNS 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 DNS module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) General Functions

	Name	Description
	TCPIP_DNS_Enable	Enables the DNS Client for the specified interface.
	TCPIP_DNS_Disable	Disables the DNS Client for the specified interface.
	TCPIP_DNS_HandlerDeRegister	Deregisters a previously registered DNS client handler.
	TCPIP_DNS_HandlerRegister	Registers a DNS client Handler.
	TCPIP_DNS_IsEnabled	Determines if the DNS client is enabled on that specified interface.
	TCPIP_DNS_IsNameResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_RemoveAll	Removes all the cached entries from DNS resolver.
	TCPIP_DNS_RemoveEntry	Remove a hostname from the DNS Hash entry.
	TCPIP_DNS_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_DNS_ClientInfoGet	Get the current DNS client parameters.
	TCPIP_DNS_EntryQuery	Queries a DNS Resolver specific entry.
	TCPIP_DNS_GetIPAddressesNumber	Get the count of resolved IPv4 and/or IPv6 address for a host name.
	TCPIP_DNS_GetIPv4Addresses	Get IPV4 addresses for a DNS resolved name.
	TCPIP_DNS_GetIPv6Addresses	Get IPV6 addresses for a DNS resolved name.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_Send_Query	Forces resolution of an address.
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the host IP address.

b) Data Types and Constants

	Name	Description
	TCPIP_DNS_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved and entry is timed out.
	TCPIP_DNS_EVENT_TYPE	This enumeration is used to notify DNS client applications.
	TCPIP_DNS_HANDLE	DNS client handle.
	TCPIP_DNS_RESULT	DNS client result codes.
	TCPIP_DNS_CLIENT_MODULE_CONFIG	Provides a place holder for DNS client configuration.
	TCPIP_DNS_CLIENT_INFO	General DNS client info.
	TCPIP_DNS_ENTRY_QUERY	DNS module query data for both IPv4 and IPv6 .
	TCPIP_DNS_ENABLE_FLAGS	Flags for enabling the DNS service on an interface.
	TCPIP_DNS_RESOLVE_TYPE	DNS query record type.

Description

This section describes the Application Programming Interface (API) functions of the DNS module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_DNS_Enable Function

Enables the DNS Client for the specified interface.

File

dns.h

C

```
bool TCPIP_DNS_Enable(TCPIP_NET_HANDLE hNet, TCPIP_DNS_ENABLE_FLAGS flags);
```

Returns

- true - if successful
- false - if unsuccessful: the requested interface could not be selected for DNS name resolving.

Description

This function enables the DNS Client name resolution for the specified interface. The additional flags give better control on how the name resolution is performed.

Remarks

The interface selection for the name resolution tries to find a valid interface, i.e. an interface that is up and has a valid DNS server. The selection is done following these rules:

- if a strict interface is set, only that interface is used for name resolution
- else if there is a preferred interface, that one will be tried first
- else the default interface is used
- else any available interface will be used

Additionally, if a retry is attempted using the same selected interface, an alternate DNS server from that interface will be selected, if available.

Only one strict interface can exist at any time. Selecting a new strict interface will replace the old one.

Only one preferred interface can exist at any time. Selecting a new preferred interface will replace the old one.

The selected interface has to be up and running for the call to succeed

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DNS Client on
flags	specify further attributes for this interface: act as a strict, preferred or default interface

Function

bool TCPIP_DNS_Enable(TCPIP_NET_HANDLE hNet, TCPIP_DNS_ENABLE_FLAGS flags)

TCPIP_DNS_Disable Function

Disables the DNS Client for the specified interface.

File

dns.h

C

```
bool TCPIP_DNS_Disable(TCPIP_NET_HANDLE hNet, bool clearCache);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DNS Client for the specified interface.

Remarks

When the DNS client is disabled on a requested interface the previously solved names will still be part of the cache and will expire when their

timeout occurs. If the TTL for a name sent by the DNS server was ignored and another default/arbitrary value was used, then the entry will stay cached until that timeout occurs (i.e. timeout not specified by the DNS server). To avoid this, you can clear the cache by setting the clearCache parameter to true.

If the disabled interface matches the strict interface set by TCPIP_DNS_Enable this function will set the strict interface to 0.

If the disabled interface matches the preferred interface set by TCPIP_DNS_Enable this function will set the preferred interface to 0.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface for which to disable the DNS Client.
clearCache	If true, all the existent name entries will be cleared from the cache

Function

bool TCPIP_DNS_Disable(TCPIP_NET_HANDLE hNet, bool clearCache)

TCPIP_DNS_HandlerDeRegister Function

Deregisters a previously registered DNS client handler.

File

dns.h

C

```
bool TCPIP_DNS_HandlerDeRegister(TCPIP_DNS_HANDLE hDns);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the DNS client event handler.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hDns	A handle returned by a previous call to TCPIP_DNS_HandlerRegister.

Function

bool TCPIP_DNS_HandlerDeRegister(TCPIP_DNS_HANDLE hDns);

TCPIP_DNS_HandlerRegister Function

Registers a DNS client Handler.

File

dns.h

C

```
TCPIP_DNS_HANDLE TCPIP_DNS_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_HANDLER handler, const
void* hParam);
```

Returns

- Returns a valid handle if the call succeeds
- Returns null handle if the call failed (out of memory, for example)

Description

This function registers a DNS client event handler. The DNS client module will call the registered handler when a DNS client event

(TCPIP_DNS_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a DNS client event occurs.
hParam	Pointer to non-volatile ASCIIZ string to be used in the handler call. It is used as a domain/host name. If not NULL, a DNS notification will be delivered only for a name resolution that matches the hParam. If the hParam == 0, then the notification is triggered for any host name resolution.

Function

TCPIP_DNS_HANDLE

TCPIP_DNS_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_HANDLER handler, const void* hParam);

TCPIP_DNS_IsEnabled Function

Determines if the DNS client is enabled on that specified interface.

File

dns.h

C

```
bool TCPIP_DNS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DNS client service is enabled on the specified interface
- false - if the DNS client service is not enabled on the specified interface

Description

This function returns the current state of DNS Client on the specified interface.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DNS_IsEnabled(CPIP_NET_HANDLE hNet)

TCPIP_DNS_IsNameResolved Function

Determines if the DNS resolution is complete and provides the host IP address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_IsNameResolved(const char* hostName, IPV4_ADDR* hostIPv4, IPV6_ADDR* hostIPv6);
```

Description

Call this function to determine if the DNS name resolution has been completed. This function allows for retrieval of separate IPv4 and IPv6 addresses for a name.

Remarks

The function will set either an IPv6 or an IPv4 address to the hostIP address, depending on what's available.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
hostIPv4	A pointer to an IPV4_ADDR structure in which to store the resolved IPv4 address if resolution is complete. Could be NULL if not needed.
hostIPv6	A pointer to an IPV6_ADDR structure in which to store the resolved IPv6 address if resolution is complete. Could be NULL if not needed.

Function

TCPIP_DNS_RESULT TCPIP_DNS_IsNameResolved(const char* hostName, IPV4_ADDR* hostIPv4, IPV6_ADDR* hostIPv6);

TCPIP_DNS_RemoveAll Function

Removes all the cached entries from DNS resolver.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_RemoveAll();
```

Returns

- TCPIP_DNS_RES_OK - If successful
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.

Description

This function is used to remove all the entries from the DNS cache. It removes both the solved and unresolved entries.

Remarks

None.

Preconditions

The DNS module must be initialized.

Function

TCPIP_DNS_RESULT TCPIP_DNS_RemoveAll(void)

TCPIP_DNS_RemoveEntry Function

Remove a hostname from the DNS Hash entry.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_RemoveEntry(const char * hostName);
```

Returns

- TCPIP_DNS_RES_OK - If name was successfully removed
- TCPIP_DNS_RES_INVALID_HOSTNAME - invalid name supplied
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.
- TCPIP_DNS_RES_NO_NAME_ENTRY - no such name exists

Description

This function is used to remove an entry (host name) from the DNS cache.

Remarks

None.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hostName	Domain name to be inserted.

Function

TCPIP_DNS_RESULT TCPIP_DNS_RemoveEntry(const char *hostName)

TCPIP_DNS_ClientTask Function

Standard TCP/IP stack module task function.

File

dns.h

C

```
void TCPIP_DNS_ClientTask();
```

Returns

None.

Description

This function performs DNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DNS module should have been initialized.

Function

void TCPIP_DNS_ClientTask(void)

TCPIP_DNS_ClientInfoGet Function

Get the current DNS client parameters.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_ClientInfoGet(TCPIP_DNS_CLIENT_INFO* pClientInfo);
```

Returns

- TCPIP_DNS_RES_OK on success
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.

Description

This function is used to get the current settings of the DNS client.

Remarks

None

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
pClientInfo	pointer to a TCPIP_DNS_CLIENT_INFO data structure to receive the DNS client information.

Function

TCPIP_DNS_RESULT TCPIP_DNS_ClientInfoGet(TCPIP_DNS_CLIENT_INFO* pClientInfo)

TCPIP_DNS_EntryQuery Function

Queries a DNS Resolver specific entry.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_EntryQuery(TCPIP_DNS_ENTRY_QUERY * pDnsQuery, int queryIndex);
```

Returns

- TCPIP_DNS_RES_OK - valid address for this index, successful

Errors:

- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.
- TCPIP_DNS_RES_INVALID_HOSTNAME - invalid string, len, pDnsQuery provided
- TCPIP_DNS_RES_EMPTY_IX_ENTRY - no name associated with this entry
- TCPIP_DNS_RES_NO_IX_ENTRY - invalid query index

Description

This function is used to query the DNS client for a specified entry. The entry to be queried is selected by its index.

Remarks

None

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
pDnsQuery	address to store the the query result
queryIndex	entry index to be selected; should start with 0

Function

TCPIP_DNS_RESULT TCPIP_DNS_EntryQuery(TCPIP_DNS_ENTRY_QUERY *pDnsQuery, int queryIndex);

TCPIP_DNS_GetIPAddressesNumber Function

Get the count of resolved IPv4 and/or IPv6 address for a host name.

File

dns.h

C

```
int TCPIP_DNS_GetIPAddressesNumber(const char* hostName, IP_ADDRESS_TYPE type);
```

Description

This function returns the total count of IPv4 and/or IPv6 addresses that exist for a resolved host name.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name
type	IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6/IP_ADDRESS_TYPE_ANY

Function

int TCPIP_DNS_GetIPAddressesNumber(const char* hostName, IP_ADDRESS_TYPE type)

TCPIP_DNS_GetIPv4Addresses Function

Get IPV4 addresses for a DNS resolved name.

File

dns.h

C

```
int TCPIP_DNS_GetIPv4Addresses(const char* hostName, int startIndex, IPV4_ADDR* pIPv4Addr, int
nIPv4Addresses);
```

Returns

> 0 - the number of addresses copied to the pIPv4Addr array 0 - if the host name was not found, invalid index, bad parameter, etc.

Description

This function will return IPv4 addresses for a host name if the DNS resolution has been completed.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name.
startIndex	starting index of the IP address to be returned when multiple addresses are available
The max number of addresses that can be stored for a host name is given by	TCPIP_DNS_CLIENT_MODULE_CONFIG::nIPv4Entries. The current number of valid entries for an address is given by TCPIP_DNS_GetIPAddressesNumber(). A valid index is [0, TCPIP_DNS_GetIPAddressesNumber(IP_ADDRESS_TYPE_IPV4));
pIPv4Addr	pointer to array of IPv4 addresses to store the host IPv4 addresses
nIPv4Addresses	number of IPv4 addresses in the pIPv4Addr array.

Function

int TCPIP_DNS_GetIPv4Addresses(const char* hostName, int startIndex, IPV4_ADDR* pIPv4Addr, int nIPv4Addresses);

TCPIP_DNS_GetIPv6Addresses Function

Get IPV6 addresses for a DNS resolved name.

File

dns.h

C

```
int TCPIP_DNS_GetIPv6Addresses(const char* hostName, int startIndex, IPV6_ADDR* pIPv6Addr, int
nIPv6Addresses);
```

Returns

> 0 - the number of addresses copied to the pIPv6Addr array 0 - if the host name was not found, invalid index, bad parameter, etc.

Description

This function will return IPv6 addresses for a host name if the DNS resolution has been completed.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name.
startIndex	starting index of the IP address to be returned when multiple addresses are available
The max number of addresses that can be stored for a host name is given by	TCPIP_DNS_CLIENT_MODULE_CONFIG::nIPv6Entries. The current number of valid entries for an address is given by TCPIP_DNS_GetIPAddressesNumber(). A valid index is [0, TCPIP_DNS_GetIPAddressesNumber(IP_ADDRESS_TYPE_IPV6));
pIPv6Addr	pointer to array of IPv6 addresses to store the host IPv6 addresses
nIPv6Addresses	number of IPv6 addresses in the pIPv6Addr array.

Function

int TCPIP_DNS_GetIPv6Addresses(const char* hostName, int startIndex, IPV6_ADDR* pIPv6Addr, int nIPv6Addresses);

TCPIP_DNS_Resolve Function

Begins resolution of an address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_Resolve(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type);
```

Returns

TCPIP_DNS_RES_OK - success, name is solved. TCPIP_DNS_RES_PENDING - operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS - name request is a IPv4 or IPv6 address

or an error code if an error occurred

Description

This function attempts to resolve a host name to an IP address. When called, it will attempt to send a DNS query to a DNS server for resolving the name. Call TCPIP_DNS_IsResolved to determine if name resolution is complete.

Remarks

To clear the cache use TCPIP_DNS_Disable(hNet, true);

Preconditions

DNS client module initialized.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
type	a TCPIP_DNS_RESOLVE_TYPE value specifying the desired resolution

Function

TCPIP_DNS_RESULT TCPIP_DNS_Resolve(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type)

TCPIP_DNS_Send_Query Function

Forces resolution of an address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_Send_Query(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type);
```

Returns

TCPIP_DNS_RES_OK - success, name is solved. TCPIP_DNS_RES_PENDING - operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS - name request is a IPv4 or IPv6 address

or an error code if an error occurred

Description

This function attempts to send a query packet for the supplied host name and DNS type.

Remarks

If the name is already part of the DNS resolution process (has been previously requested with TCPIP_DNS_Resolve or TCPIP_DNS_Send_Query) the function will force a new DNS query . If the name resolution is already solved and completed, this function will mark it as incomplete and a new response from the server will be requested.

If the name was not part of the DNS client resolution, then this function is equivalent to TCPIP_DNS_Resolve().

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
type	a TCPIP_DNS_RESOLVE_TYPE value specifying the desired resolution

Function

TCPIP_DNS_RESULT TCPIP_DNS_Send_Query(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type)

TCPIP_DNS_IsResolved Function

Determines if the DNS resolution is complete and provides the host IP address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_IsResolved(const char* hostName, IP_MULTI_ADDRESS* hostIP, IP_ADDRESS_TYPE type);
```

Description

Call this function to determine if the DNS resolution of an address has been completed. If so, the resolved address will be provided in hostIP.

Remarks

The function will set either an IPv6 or an IPv4 address to the hostIP address, depending on what's available.

If type IP_ADDRESS_TYPE_ANY is specified the hostIP will be updated with the first available solved address: either IPv6 or IPv4!

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
hostIP	A pointer to an IP_MULTI_ADDRESS structure in which to store the resolved IPv4/IPv6 address if resolution is complete. Could be NULL if not needed.
type	type of address needed: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6/IP_ADDRESS_TYPE_ANY

Function

TCPIP_DNS_RESULT TCPIP_DNS_IsResolved(const char* hostName, IP_MULTI_ADDRESS* hostIP, IP_ADDRESS_TYPE type)

b) Data Types and Constants

TCPIP_DNS_EVENT_HANDLER Type

Notification handler that can be called when a specific entry is resolved and entry is timed out.

File

dns.h

C

```
typedef void (* TCPIP_DNS_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_TYPE evType, const char*
name, const void* param);
```

Description

Type: TCPIP_DNS_EVENT_HANDLER

The format of a notification handler registered with the DNS module. Once an DNS event occurs the DNS service will be called for the registered handler.

Remarks

If pNetIf == 0, the notification is called for events on any interface.

Parameters

Parameters	Description
hNet	the interface on which the DNS event occurred
evType	the DNS reported event
name	the host name associated with the event
param	additional user parameter - see TCPIP_DNS_HandlerRegister

TCPIP_DNS_EVENT_TYPE Enumeration

This enumeration is used to notify DNS client applications.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_EVENT_NONE,
  TCPIP_DNS_EVENT_NAME_QUERY,
  TCPIP_DNS_EVENT_NAME_RESOLVED,
  TCPIP_DNS_EVENT_NAME_EXPIRED,
  TCPIP_DNS_EVENT_NAME_REMOVED,
  TCPIP_DNS_EVENT_NAME_ERROR,
  TCPIP_DNS_EVENT_SOCKET_ERROR,
  TCPIP_DNS_EVENT_NO_INTERFACE
} TCPIP_DNS_EVENT_TYPE;
```

Members

Members	Description
TCPIP_DNS_EVENT_NONE	DNS no event
TCPIP_DNS_EVENT_NAME_QUERY	DNS Query sent
TCPIP_DNS_EVENT_NAME_RESOLVED	DNS Name resolved
TCPIP_DNS_EVENT_NAME_EXPIRED	name entry expired
TCPIP_DNS_EVENT_NAME_REMOVED	name removed to make room for another entry
TCPIP_DNS_EVENT_NAME_ERROR	no such name reported by the DNS server
TCPIP_DNS_EVENT_SOCKET_ERROR	a DNS probe could not be sent, socket was busy, TX failed, etc.
TCPIP_DNS_EVENT_NO_INTERFACE	a DNS probe could not be sent, no DNS interface could be selected

Description

Enumeration: TCPIP_DNS_EVENT_TYPE

These events are used while notifying to the registered applications.

Remarks

None.

TCPIP_DNS_HANDLE Type

DNS client handle.

File

dns.h

C

```
typedef const void* TCPIP_DNS_HANDLE;
```

Description

Type: TCPIP_DNS_HANDLE

A handle that a application needs to use when deregistering a notification handler.

Remarks

This handle can be used by the application after the event handler has been registered.

TCPIP_DNS_RESULT Enumeration

DNS client result codes.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_RES_OK = 0,
  TCPIP_DNS_RES_PENDING,
  TCPIP_DNS_RES_NAME_IS_IPADDRESS,
  TCPIP_DNS_RES_NO_NAME_ENTRY = -1,
  TCPIP_DNS_RES_NO_IP_ENTRY = -2,
  TCPIP_DNS_RES_NO_IX_ENTRY = -3,
  TCPIP_DNS_RES_EMPTY_IX_ENTRY = -4,
  TCPIP_DNS_RES_SERVER_TMO = -5,
  TCPIP_DNS_RES_NO_SERVICE = -6,
  TCPIP_DNS_RES_NO_INTERFACE = -7,
  TCPIP_DNS_RES_CACHE_FULL = -8,
  TCPIP_DNS_RES_INVALID_HOSTNAME = -9,
  TCPIP_DNS_RES_SOCKET_ERROR = -10
} TCPIP_DNS_RESULT;
```

Members

Members	Description
TCPIP_DNS_RES_OK = 0	operation succeeded
TCPIP_DNS_RES_PENDING	operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS	DNS request is a IPv4 or IPv6 address
TCPIP_DNS_RES_NO_NAME_ENTRY = -1	no such name exists
TCPIP_DNS_RES_NO_IP_ENTRY = -2	no such IP type exists
TCPIP_DNS_RES_NO_IX_ENTRY = -3	no such index exists
TCPIP_DNS_RES_EMPTY_IX_ENTRY = -4	no entry associated to this index exists
TCPIP_DNS_RES_SERVER_TMO = -5	DNS server response tmo
TCPIP_DNS_RES_NO_SERVICE = -6	DNS service not implemented or uninitialized
TCPIP_DNS_RES_NO_INTERFACE = -7	no interface for DNS traffic available
TCPIP_DNS_RES_CACHE_FULL = -8	the cache is full and no entry could be added
TCPIP_DNS_RES_INVALID_HOSTNAME = -9	Invalid hostname
TCPIP_DNS_RES_SOCKET_ERROR = -10	DNS UDP socket error: not ready, TX error, etc.

Description

Enumeration: TCPIP_DNS_RESULT

DNS Client operations results.

Remarks

None.

TCPIP_DNS_CLIENT_MODULE_CONFIG Structure

Provides a place holder for DNS client configuration.

File

dns.h

C

```
typedef struct {
  bool deleteOldLease;
  int cacheEntries;
  uint32_t entrySolvedTmo;
  int nIPv4Entries;
  IP_ADDRESS_TYPE ipAddressType;
  int nIPv6Entries;
} TCPIP_DNS_CLIENT_MODULE_CONFIG;
```

Members

Members	Description
bool deleteOldLease;	Delete old cache if still in place, specific DNS parameters
int cacheEntries;	Max number of cache entries
uint32_t entrySolvedTmo;	Solved entry removed after this tmo if not referenced - seconds
int nIPv4Entries;	Number of IPv4 entries per DNS name and Default value is 1.
IP_ADDRESS_TYPE ipAddressType;	IP protocol type to use for connecting to the DNS server: IPv4 or IPv6 Currently only IPv4 is supported and this parameter is not used Reserved for future improvements
int nIPv6Entries;	Number of IPv6 address per DNS Name Default value is 1 and is used only when IPv6 is enabled

Description

Structure: TCPIP_DNS_CLIENT_MODULE_CONFIG

DNS module runtime configuration and initialization parameters.

Remarks

None.

TCPIP_DNS_CLIENT_INFO Structure

General DNS client info.

File

dns.h

C

```
typedef struct {
  TCPIP_NET_HANDLE strictNet;
  TCPIP_NET_HANDLE prefNet;
  uint32_t dnsTime;
  uint16_t pendingEntries;
  uint16_t currentEntries;
  uint16_t totalEntries;
} TCPIP_DNS_CLIENT_INFO;
```

Members

Members	Description
TCPIP_NET_HANDLE strictNet;	current strict DNS interface

TCPIP_NET_HANDLE prefNet;	current preferred DNS interface
uint32_t dnsTime;	current DNS time, seconds
uint16_t pendingEntries;	number of entries that need to be solved
uint16_t currentEntries;	number of solved and unslolved name entries
uint16_t totalEntries;	total number of supported name entries

Description

Structure: TCPIP_DNS_CLIENT_INFO

DNS module data structure used for getting information about the module settings.

Remarks

None.

TCPIP_DNS_ENTRY_QUERY Structure

DNS module query data for both IPv4 and IPv6 .

File

dns.h

C

```
typedef struct {
  char* hostName;
  int nameLen;
  IPV4_ADDR * ipv4Entry;
  int nIPv4Entries;
  IPV6_ADDR * ipv6Entry;
  int nIPv6Entries;
  TCPIP_DNS_RESULT status;
  uint32_t ttlTime;
  TCPIP_NET_HANDLE hNet;
  int serverIx;
  int nIPv4ValidEntries;
  int nIPv6ValidEntries;
} TCPIP_DNS_ENTRY_QUERY;
```

Members

Members	Description
char* hostName;	Pointer to a name to receive the host name for that particular entry
int nameLen;	hostName buffer size
IPV4_ADDR * ipv4Entry;	array of IPv4 entries/addresses to be populated
int nIPv4Entries;	number of entries in the ipv4Entry[] array;
IPV6_ADDR * ipv6Entry;	array of IPv6 entries/addresses to be populated
int nIPv6Entries;	number of entries in the ipv6Entry[] array;
TCPIP_DNS_RESULT status;	current status for this name: • TCPIP_DNS_RES_OK: name is resolved • TCPIP_DNS_RES_PENDING: name is pending • TCPIP_DNS_RES_SERVER_TMO: server timeout
uint32_t ttlTime;	time to live for a solved DNS entry
TCPIP_NET_HANDLE hNet;	interface the name was obtained or on which the query is currently ongoing
int serverIx;	index of the server used on that interface
int nIPv4ValidEntries;	number of valid entries written to the ipv4Entry
int nIPv6ValidEntries;	number of valid entries written to the ipv6Entry

Description

Structure: TCPIP_DNS_ENTRY_QUERY

DNS module uses this structure to return information about a resolved IPv4 and IPv6 address.

Remarks

None.

TCPIP_DNS_ENABLE_FLAGS Enumeration

Flags for enabling the DNS service on an interface.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_ENABLE_DEFAULT = 0,
  TCPIP_DNS_ENABLE_STRICT,
  TCPIP_DNS_ENABLE_PREFERRED
} TCPIP_DNS_ENABLE_FLAGS;
```

Members

Members	Description
TCPIP_DNS_ENABLE_DEFAULT = 0	the interface is capable of performing DNS name resolution
TCPIP_DNS_ENABLE_STRICT	only this interface will be used for DNS name resolution
TCPIP_DNS_ENABLE_PREFERRED	prefer this interface when doing DNS name resolution

Description

Enumeration: TCPIP_DNS_ENABLE_FLAGS

This enumeration lists the TCPIP_DNS_ENABLE_FLAGS argument for TCPIP_DNS_Enable.

Remarks

See the TCPIP_DNS_Enable description for details

TCPIP_DNS_RESOLVE_TYPE Enumeration

DNS query record type.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_TYPE_A = 1,
  TCPIP_DNS_TYPE_MX = 15,
  TCPIP_DNS_TYPE_AAAA = 28u,
  TCPIP_DNS_TYPE_ANY = 0xff
} TCPIP_DNS_RESOLVE_TYPE;
```

Members

Members	Description
TCPIP_DNS_TYPE_A = 1	Indicates an A (standard address) record.
TCPIP_DNS_TYPE_MX = 15	Indicates an MX (mail exchanger) record.
TCPIP_DNS_TYPE_AAAA = 28u	Indicates a quad-A (IPv6 address) address record.

Description

Enumeration: TCPIP_DNS_RESOLVE_TYPE

This enumeration lists the RecordType argument for TCPIP_DNS_Resolve. The stack supports DNS_TYPE_A and DNS_TYPE_AAAA.

Remarks

None.

Files

Files

Name	Description
dns.h	DNS definitions and interface file.

dns_config.h	DNS configuration file

Description

This section lists the source and header files used by the library.

dns.h

DNS definitions and interface file.

Enumerations

	Name	Description
	TCPIP_DNS_ENABLE_FLAGS	Flags for enabling the DNS service on an interface.
	TCPIP_DNS_EVENT_TYPE	This enumeration is used to notify DNS client applications.
	TCPIP_DNS_RESOLVE_TYPE	DNS query record type.
	TCPIP_DNS_RESULT	DNS client result codes.

Functions

	Name	Description
	TCPIP_DNS_ClientInfoGet	Get the current DNS client parameters.
	TCPIP_DNS_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_DNS_Disable	Disables the DNS Client for the specified interface.
	TCPIP_DNS_Enable	Enables the DNS Client for the specified interface.
	TCPIP_DNS_EntryQuery	Queries a DNS Resolver specific entry.
	TCPIP_DNS_GetIPAddressesNumber	Get the count of resolved IPv4 and/or IPv6 address for a host name.
	TCPIP_DNS_GetIPv4Addresses	Get IPV4 addresses for a DNS resolved name.
	TCPIP_DNS_GetIPv6Addresses	Get IPV6 addresses for a DNS resolved name.
	TCPIP_DNS_HandlerDeRegister	Deregisters a previously registered DNS client handler.
	TCPIP_DNS_HandlerRegister	Registers a DNS client Handler.
	TCPIP_DNS_IsEnabled	Determines if the DNS client is enabled on that specified interface.
	TCPIP_DNS_IsNameResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_RemoveAll	Removes all the cached entries from DNS resolver.
	TCPIP_DNS_RemoveEntry	Remove a hostname from the DNS Hash entry.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_Send_Query	Forces resolution of an address.

Structures

	Name	Description
	TCPIP_DNS_CLIENT_INFO	General DNS client info.
	TCPIP_DNS_CLIENT_MODULE_CONFIG	Provides a place holder for DNS client configuration.
	TCPIP_DNS_ENTRY_QUERY	DNS module query data for both IPv4 and IPv6 .

Types

	Name	Description
	TCPIP_DNS_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved and entry is timed out.
	TCPIP_DNS_HANDLE	DNS client handle.

Description

Domain Name System (DNS) Client API Header file

This source file contains the DNS client module API.

File Name

dns.h

Company

Microchip Technology Inc.

dns_config.h

DNS configuration file

Macros

	Name	Description
	TCPIP_DNS_CLIENT_ADDRESS_TYPE	This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development
	TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL	Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0
	TCPIP_DNS_CLIENT_CACHE_ENTRIES	Number of DNS resolver entries
	TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO	DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO	Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO
	TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO	Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO
	TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN	Max DNS host name size
	TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES	Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1: • the default interface should always be considered for DNS resolution Depending on how many active interfaces select those to be considered for DNS resolution
	TCPIP_DNS_CLIENT_SERVER_TMO	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds
	TCPIP_DNS_CLIENT_TASK_PROCESS_RATE	DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNS_CLIENT_USER_NOTIFICATION	allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used

Description

Domain Name Service (CNS) Configuration file

This file contains the DNS module configuration options

File Name

dns_config.h

Company

Microchip Technology Inc.

DNS Server Module

This section describes the TCP/IP Stack Library DNS Server module.

Introduction

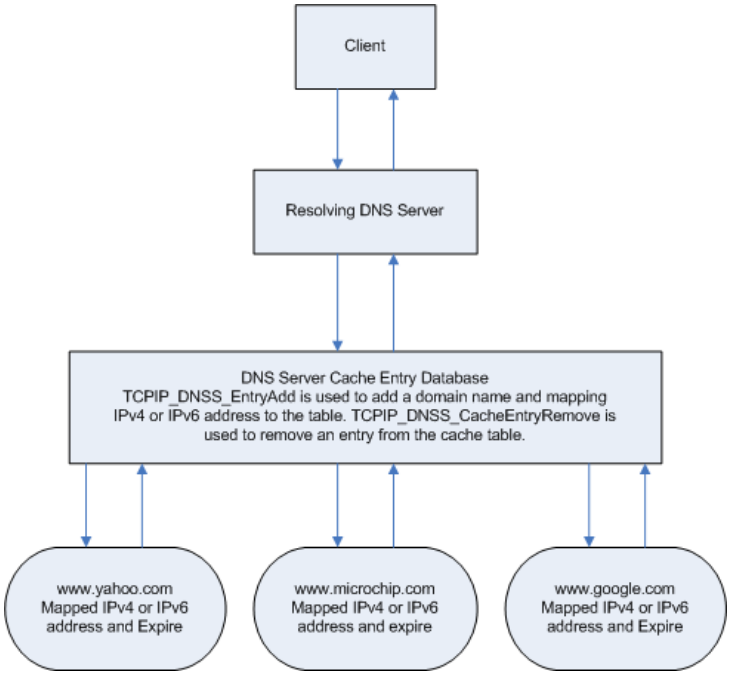
TCP/IP Stack Library Domain Name System Server (DNSS) Module for Microchip Microcontrollers

This library provides the API of the DNS Server module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

A step-by-step analysis of the DNS Server is as follows:

1. DNS server can be enabled or disabled by using the dnss command (it is not interface based).
2. The client initiates a query to find the Domain Name Server. The client sends the query to the DNS Server of the TCP/IP stack.
3. The DNS Server of the TCP/IP Stack first checks the Boolean value of replyBoardAddr of DNSS_MODULE_CONFIG. If this is true, the DNS server will send an answer with its own board address as the DNS server address to the requested Client. If it is false, it then checks its own cache, whether or not it already knows the answer. But, as the answer is not present, it generates NO response for the client query.
   - Cache entry is updated by adding a entry to it using command handler
   - Cache entry is the table of Name , IPv4/IPv6 address. and expire time
   - The dnsserv command is used to add, delete and display the DNS server details
   - Users can add either one IPv4 or IPv6 address at a time
   - Maximum Name and IPv4/IPv6 address combination is determined from the IPv4EntriesPerDNSName and IPv6EntriesPerDNSName of DNSS_MODULE_CONFIG
4. Authoritative Name servers and Addition Name servers are not processed.
5. The TCP/IP DNS Server stores the answer in its cache for future use and answers to the client by sending the IP address of the Domain Name Server, which is added by the user using command dnsserv.
6. The client may store the answer to the DNS query in it's own cache for future use. Then, the client communicates directly with the Domain Name server using the IP address.



Using the Library

This topic describes the basic architecture of the DNSS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dnss.h`

The interface to the DNS TCP/IP Stack library is defined in the `dnss.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DNS TCP/IP Stack library should include `tcpip.h`.

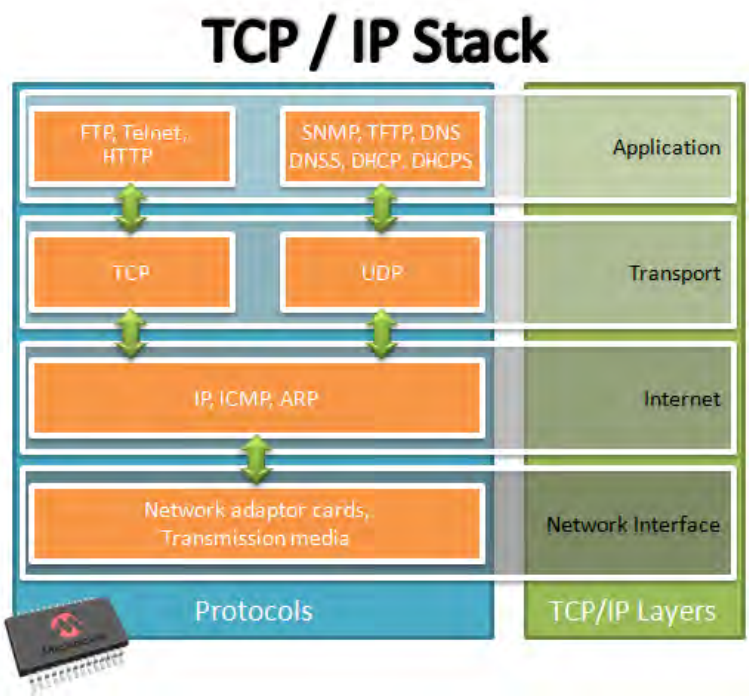
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

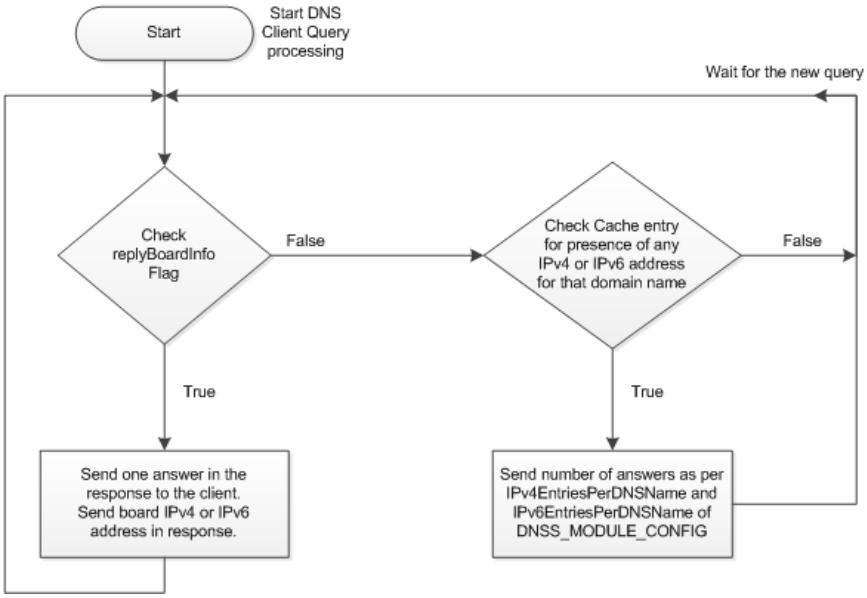
This library provides the API of the DNSS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DNSS is part of the Application Layer, as illustrated in the following figure.



DNSS resolution operations follow a simple state machine, as indicated in the following 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 DNSS module.

Library Interface Section	Description
General Functions	This section provides general interface routines to DNS

Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES	Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.
	TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.
	TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.
	TCPIP_DNSS_HOST_NAME_LEN	Default DNS host name length
	TCPIP_DNSS_REPLY_BOARD_ADDR	Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1
	TCPIP_DNSS_TASK_PROCESS_RATE	DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNSS_TTL_TIME	Default TTL time for a IP address is 10 minutes

Description

The configuration of the DNSS TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DNSS TCP/IP Stack. Based on the selections made, the DNSS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DNSS TCP/IP Stack.

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.

TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES
(TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS+TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS)
```

Description

Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.

TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS 2
```

Description

Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.

TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS 1
```

Description

Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.

TCPIP_DNSS_HOST_NAME_LEN Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_HOST_NAME_LEN 64u
```

Description

Default DNS host name length

TCPIP_DNSS_REPLY_BOARD_ADDR Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_REPLY_BOARD_ADDR 1
```

Description

Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1

TCPIP_DNSS_TASK_PROCESS_RATE Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_TASK_PROCESS_RATE (33)
```

Description

DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DNSS_TTL_TIME Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_TTL_TIME (10*60)
```

Description

Default TTL time for a IP address is 10 minutes

Building the Library

This section lists the files that are available in the DNS Server module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/dnss.c`	DNS Server 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 DNS Server module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) General Functions

	Name	Description
	TCPIP_DNSS_AddressCntGet	Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index
	TCPIP_DNSS_CacheEntryRemove	Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address
	TCPIP_DNSS_Disable	Disables the DNS Server for the specified interface.
	TCPIP_DNSS_Enable	Enables the DNS server for the specified interface.
	TCPIP_DNSS_EntryAdd	Add a IPv4 or IPv6 entry to the DNS server table.
	TCPIP_DNSS_IsEnabled	Determines if the DNS Server is enabled on the specified interface.
	TCPIP_DNSS_EntryGet	Get DNS server IP address details from resolver pool entry.
	TCPIP_DNSS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_DNSS_MODULE_CONFIG	Provides a place holder for DNS server configuration.
	TCPIP_DNSS_RESULT	DNS result codes

Description

This section describes the Application Programming Interface (API) functions of the DNSS module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_DNSS_AddressCntGet Function

Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_AddressCntGet(int index, uint8_t * hostName, uint8_t * ipCount);
```

Returns

TCPIP_DNSS_RESULT

Description

This function is used to get the number of IPv4 and IPv6 address entry details from hash entry table. Here input parameter is index value. Output will be the hostname and IP address count. IP address count is the summation of both IPv4 and IPv6 address.

Remarks

None.

Preconditions

DNSServerInit should be called.

Parameters

Parameters	Description
hostName	Get DNS host name
ipCount	Get the number of IPv4 or IPv6 Server address
index	Server count details per index value

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_AddressCntGet(uint8_t * hostName,uint8_t * ipCount,int index)

TCPIP_DNSS_CacheEntryRemove Function

Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_CacheEntryRemove(const char* name, IP_ADDRESS_TYPE type, IP_MULTI_ADDRESS*
pAdd);
```

Returns

- DNSS_RES_OK - If entry remove is successful
- DNSS_RES_NO_ENTRY - No such entry is present
- DNSS_RES_MEMORY_FAIL - No Memory is present for IPv4 or IPv6 address type

Description

This function is used for command prompt dnsDelSrv and to delete a entry IPv4 or IPv6 from hostName. Every time this is used to delete either one IPv4 or IPv6 address with respect to the DNS hostname, there will be an expire time for a Host entry.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
name	Hostname
type	IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6
pAdd	v4 or v6 address

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_CacheEntryRemove(const char* name, IP_ADDRESS_TYPE type, IP_MULTI_ADDRESS* pAdd)

TCPIP_DNSS_Disable Function

Disables the DNS Server for the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DNS Server for the specified interface. If it is already disabled, no action is taken.

Remarks

none

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to disable the DNS Server.

Function

bool TCPIP_DNSS_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_Enable Function

Enables the DNS server for the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function enables the DNS Server for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DNS Server on.

Function

bool TCPIP_DNSS_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_EntryAdd Function

Add a IPv4 or IPv6 entry to the DNS server table.

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_EntryAdd(const char* name, IP_ADDRESS_TYPE type, IP_MULTI_ADDRESS* pAdd,
uint32_t validStartTime);
```

Returns

- DNSS_RES_OK - If entry remove is successful
- DNSS_RES_NO_ENTRY - If Hostname is NULL value and Invalid IP type
- DNSS_RES_MEMORY_FAIL - No Memory is present for IPv4 or IPv6 address type
- DNSS_RES_CACHE_FULL - If there is no space for the new entry

Description

This function is used to add a IPv4 or IPv6 entry. Every time this is used to add either one IPv4 or IPv6 address with respect to the DNS hostname, there will be an expire time for a Host entry.

Remarks

None.

Preconditions

DNS Server Initialized.

Parameters

Parameters	Description
name	Hostname
type	IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6
pAdd	IPv4 or IPv6 address
validStartTime	Time-out value for the entry

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_EntryAdd(const char* name, IP_ADDRESS_TYPE type,
IP_MULTI_ADDRESS* pAdd,uint32_t validStartTime);

TCPIP_DNSS_IsEnabled Function

Determines if the DNS Server is enabled on the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DNS Server is enabled on the specified interface
- false - if the DNS Server is not enabled on the specified interface

Description

This function returns the current state of the DNS Server on the specified interface.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DNSS_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_EntryGet Function

Get DNS server IP address details from resolver pool entry.

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_EntryGet(uint8_t * hostName, IP_ADDRESS_TYPE type, int index,
IP_MULTI_ADDRESS* pGetAdd, uint32_t * ttltime);
```

Returns

TCPIP_DNSS_RESULT

Description

This function is used to get the DNS server IPv4 or IPv6 address from resolver pool entry as per hostname and IP type and from the index value. This IP type can be a IPv4 and IPv6 type. This is used for DNS record type.

Remarks

None.

Preconditions

DNSServerInit should be called.

Parameters

Parameters	Description
hostName	DNS host name
type	DNS IP type (it will be ipv4 or ipv6 type and it is mapped to DNS record type) this can be used for DNS record type
index	get the next entry after this index value
pGetAdd	get the zero th IPv4 or IPv6 address
ttltime	timeout value

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_EntryGet(uint8_t * hostName,IP_ADDRESS_TYPE type,
int index, IP_MULTI_ADDRESS* pGetAdd,uint32_t *ttltime);

TCPIP_DNSS_Task Function

Standard TCP/IP stack module task function.

File

dnss.h

C

```
void TCPIP_DNSS_Task();
```

Returns

None.

Description

This function performs DNSS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DNSS module should have been initialized.

Function

void TCPIP_DNSS_Task(void)

b) Data Types and Constants

TCPIP_DNSS_MODULE_CONFIG Structure

Provides a place holder for DNS server configuration.

File

dnss.h

C

```
typedef struct {
  bool deleteOldLease;
  bool replyBoardAddr;
  size_t IPv4EntriesPerDNSName;
  size_t IPv6EntriesPerDNSName;
} TCPIP_DNSS_MODULE_CONFIG;
```

Members

Members	Description
bool deleteOldLease;	Delete old cache if still in place,
bool replyBoardAddr;	Reply with board address specific DNS server parameters
size_t IPv4EntriesPerDNSName;	Number of IPv4 entries per DNS name. Default value is 1.
size_t IPv6EntriesPerDNSName;	Number of IPv6 address per DNS Name. Default value is 1 Used only when IPv6 is enabled

Description

Structure: TCPIP_DNSS_MODULE_CONFIG

DNS Server run-time configuration and initialization parameter.

Remarks

None.

TCPIP_DNSS_RESULT Enumeration

DNS result codes

File

dnss.h

C

```
typedef enum {
  DNSS_RES_OK = 0,
  DNSS_RES_NO_SERVICE = -1,
  DNSS_RES_CACHE_FULL = -2,
  DNSS_RES_NO_ENTRY = -3,
  DNSS_RES_NO_IPADDRESS = -4,
  DNSS_RES_MEMORY_FAIL = -5,
```

```
  DNSS_RES_DUPLICATE_ENTRY = -6
} TCPIP_DNSS_RESULT;
```

Members

Members	Description
DNSS_RES_OK = 0	operation succeeded
DNSS_RES_NO_SERVICE = -1	DNS service not implemented or uninitialized
DNSS_RES_CACHE_FULL = -2	the cache is full and no entry could be added
DNSS_RES_NO_ENTRY = -3	DNSS no such name
DNSS_RES_NO_IPADDRESS = -4	No such IP address
DNSS_RES_MEMORY_FAIL = -5	out of memory failure
DNSS_RES_DUPLICATE_ENTRY = -6	duplicate entry was found

Description

Enumeration: TCPIP_DNSS_RESULT

DNS Server operations results.

Remarks

None.

Files

Files

Name	Description
dnss.h	DNS server definitions and interface file
dnss_config.h	DNSS configuration file

Description

This section lists the source and header files used by the library.

dnss.h

DNS server definitions and interface file

Enumerations

	Name	Description
	TCPIP_DNSS_RESULT	DNS result codes

Functions

	Name	Description
	TCPIP_DNSS_AddressCntGet	Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index
	TCPIP_DNSS_CacheEntryRemove	Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address
	TCPIP_DNSS_Disable	Disables the DNS Server for the specified interface.
	TCPIP_DNSS_Enable	Enables the DNS server for the specified interface.
	TCPIP_DNSS_EntryAdd	Add a IPv4 or IPv6 entry to the DNS server table.
	TCPIP_DNSS_EntryGet	Get DNS server IP address details from resolver pool entry.
	TCPIP_DNSS_IsEnabled	Determines if the DNS Server is enabled on the specified interface.
	TCPIP_DNSS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DNSS_MODULE_CONFIG	Provides a place holder for DNS server configuration.

Description

Domain Name System (DNS) server Header file

This source file contains the DNS Server module API

File Name

dnss.h

Company

Microchip Technology Inc.

dnss_config.h

DNSS configuration file

Macros

	Name	Description
	TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES	Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.
	TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.
	TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.
	TCPIP_DNSS_HOST_NAME_LEN	Default DNS host name length
	TCPIP_DNSS_REPLY_BOARD_ADDR	Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1
	TCPIP_DNSS_TASK_PROCESS_RATE	DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNSS_TTL_TIME	Default TTL time for a IP address is 10 minutes

Description

Domain Name System Server (DNSS) Configuration file

This file contains the DNSS module configuration options

File Name

dnss_config.h

Company

Microchip Technology Inc.

Dynamic DNS Module

This section describes the TCP/IP Stack Library Dynamic DNS (DDNS) module.

Introduction

TCP/IP Stack Library Dynamic Domain Name System (DDNS) Module for Microchip Microcontrollers

This library provides the API of the DDNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

Note that this only solves one of the two problems for communicating to devices on local subnets from the Internet. While Dynamic DNS can help to locate the device, the router or firewall it sits behind must still properly forward the incoming connection request. This generally requires port forwarding to be configured for the router behind which the device is located.

The Dynamic DNS client supports the popular interface used by No-IP (noip.com) and DNS-O-Matic (dnsomatic.com).

Important! The dynamic DNS services stipulate that updates should be made no more frequently than 10 minutes, and only when the IP address has changed. Updates made more often than that are considered abusive, and may eventually cause your account to be disabled. Production devices that get rebooted frequently may need to store the last known IP in non-volatile memory. You also should not enable this module while testing the rest of your application.

Using the Library

This topic describes the basic architecture of the DDNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ddns.h`

The interface to the DDNS TCP/IP Stack library is defined in the `ddns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DDNS TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the DDNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DDNS 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 Dynamic DNS module.

Library Interface Section	Description
Functions	Routines for Configuring DDNS
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	TCPIP_DDNS_TASK_TICK_RATE	dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the DDNS TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DDNS TCP/IP Stack Library. Based on the selections made, the DDNS TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the DDNS TCP/IP Stack 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.

TCPIP_DDNS_CHECKIP_SERVER Macro

File

ddns_config.h

C

```
#define TCPIP_DDNS_CHECKIP_SERVER (const uint8_t*)"checkip.dyndns.com"
```

Description

Default CheckIP server for determining current IP address

TCPIP_DDNS_TASK_TICK_RATE Macro

File

ddns_config.h

C

```
#define TCPIP_DDNS_TASK_TICK_RATE 777
```

Description

dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the Dynamic DNS module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/ddns.c`	Dynamic DNS Server 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 Dynamic DNS module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a preconfigured Dynamic DNS service.
	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update.
	TCPIP_DDNS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Description

This section describes the Application Programming Interface (API) functions of the Dynamic DNS module.

Refer to each section for a detailed description.

a) Functions

TCPIP_DDNS_LastIPGet Function

Returns the status of the most recent update.

File

ddns.h

C

```
IPV4_ADDR TCPIP_DDNS_LastIPGet();
```

Returns

DDNS_STATUS indicating the status code for the most recent update.

Description

This function returns the status of the most recent update. See the DDNS_STATUS enumeration for possible codes.

Preconditions

None.

Function

DDNS_STATUS TCPIP_DDNS_LastStatusGet(void)

TCPIP_DDNS_LastStatusGet Function

Returns the last known external IP address of the device.

File

ddns.h

C

```
DDNS_STATUS TCPIP_DDNS_LastStatusGet();
```

Returns

The last known external IP address of the device.

Description

This function returns the last known external IP address of the device.

Preconditions

None.

Function

IPV4_ADDR TCPIP_DDNS_LastIPGet(void)

TCPIP_DDNS_ServiceSet Function

Selects a preconfigured Dynamic DNS service.

File

ddns.h

C

```
void TCPIP_DDNS_ServiceSet(DDNS_SERVICES svc);
```

Returns

None.

Description

This function selects a Dynamic DNS service based on parameters configured in ddnsServiceHosts and ddnsServicePorts. These arrays must match the DDNS_SERVICES enumeration.

Preconditions

None.

Parameters

Parameters	Description
svc	one of the DDNS_SERVICES elements to indicate the selected service

Function

void TCPIP_DDNS_ServiceSet(DDNS_SERVICES svc)

TCPIP_DDNS_UpdateForce Function

Forces an immediate DDNS update.

File

ddns.h

C

```
void TCPIP_DDNS_UpdateForce();
```

Returns

None.

Description

This function forces the DDNS Client to execute a full update immediately. Any error message is cleared, and the update will be executed whether the IP address has changed or not. Call this function every time the DDNSClient parameters have been modified.

Preconditions

TCPIP_DDNS_Initialize must have been called.

Function

void TCPIP_DDNS_UpdateForce(void)

TCPIP_DDNS_Task Function

Standard TCP/IP stack module task function.

File

ddns.h

C

```
void TCPIP_DDNS_Task();
```

Returns

None.

Description

Performs DDNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DDNS module should have been initialized

Function

void TCPIP_DDNS_Task(void)

b) Data Types and Constants

DDNS_MODULE_CONFIG Structure

File

ddns.h

C

```
typedef struct {
} DDNS_MODULE_CONFIG;
```

Description

This is type DDNS_MODULE_CONFIG.

DDNS_POINTERS Structure

Configuration parameters for the Dynamic DNS Client

File

ddns.h

C

```
typedef struct {
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } CheckIPServer;
  uint16_t CheckIPPort;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } UpdateServer;
  uint16_t UpdatePort;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Username;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Password;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Host;
  struct {
    unsigned char CheckIPServer : 1;
    unsigned char UpdateServer : 1;
    unsigned char Username : 1;
    unsigned char Password : 1;
    unsigned char Host : 1;
  } ROMPointers;
} DDNS_POINTERS;
```

Description

This structure of pointers configures the Dynamic DNS Client. Initially, all pointers will be null and the client will be disabled. `Set DDNSClient.[field name].szRAM` to use a string stored in RAM, or `DDNSClient.[field name].szROM` to use a string stored in const. Where, `[field name]`, is one of the following parameters.

If a const string is specified, `DDNSClient.ROMPointers.[field name]` must also be set to 1 to indicate that this field should be retrieved from const instead of RAM.

Parameters

Parameters	Description
CheckIPServer	The server used to determine the external IP address
CheckIPPort	Port on the above server to connect to
UpdateServer	The server where updates should be posted
UpdatePort	Port on the above server to connect to
Username	The user name for the dynamic DNS server
Password	The password to supply when making updates
Host	The host name you wish to update
ROMPointers	Indicates which parameters to read from const instead of RAM.

DDNS_SERVICES Enumeration

File

ddns.h

C

```
typedef enum {
  DYNDNS_ORG = 0u,
  NO_IP_COM,
  DNSOMATIC_COM
} DDNS_SERVICES;
```

Members

Members	Description
DYNDNS_ORG = 0u	www.dyndns.org
NO_IP_COM	www.no-ip.com
DNSOMATIC_COM	www.dnsomatic.com

Description

Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.

DDNS_STATUS Enumeration

File

ddns.h

C

```
typedef enum {
  DDNS_STATUS_GOOD = 0u,
  DDNS_STATUS_NOCHG,
  DDNS_STATUS_ABUSE,
  DDNS_STATUS_BADSYS,
  DDNS_STATUS_BADAGENT,
  DDNS_STATUS_BADAUTH,
  DDNS_STATUS_NOT_DONATOR,
  DDNS_STATUS_NOT_FQDN,
  DDNS_STATUS_NOHOST,
  DDNS_STATUS_NOT_YOURS,
  DDNS_STATUS_NUMHOST,
  DDNS_STATUS_DNSERR,
  DDNS_STATUS_911,
  DDNS_STATUS_UPDATE_ERROR,
  DDNS_STATUS_UNCHANGED,
  DDNS_STATUS_CHECKIP_ERROR,
  DDNS_STATUS_DNS_ERROR,
  DDNS_STATUS_SKT_ERROR,
  DDNS_STATUS_INVALID,
  DDNS_STATUS_UNKNOWN
} DDNS_STATUS;
```

Members

Members	Description
DDNS_STATUS_GOOD = 0u	Update successful, hostname is now updated
DDNS_STATUS_NOCHG	Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.
DDNS_STATUS_ABUSE	The hostname specified is blocked for update abuse.
DDNS_STATUS_BADSYS	System parameter not valid. Should be dyndns, statdns or custom.
DDNS_STATUS_BADAGENT	The user agent was blocked or not sent.
DDNS_STATUS_BADAUTH	The username and password pair do not match a real user.
DDNS_STATUS_NOT_DONATOR	An option available only to credited users (such as offline URL) was specified, but the user is not a credited user. If multiple hosts were specified, only a single !donator will be returned.
DDNS_STATUS_NOT_FQDN	The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).
DDNS_STATUS_NOHOST	The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).
DDNS_STATUS_NOT_YOURS	The hostname specified does not belong to this user account.
DDNS_STATUS_NUMHOST	Too many hosts specified in an update.
DDNS_STATUS_DNSERR	Unspecified DNS error encountered by the DDNS service.
DDNS_STATUS_911	There is a problem or scheduled maintenance with the DDNS service.
DDNS_STATUS_UPDATE_ERROR	Error communicating with Update service.
DDNS_STATUS_UNCHANGED	The IP Check indicated that no update was necessary.
DDNS_STATUS_CHECKIP_ERROR	Error communicating with CheckIP service.
DDNS_STATUS_DNS_ERROR	DNS error resolving the CheckIP service.
DDNS_STATUS_SKT_ERROR	TCP socket opening error

DDNS_STATUS_INVALID	DDNS Client data is not valid.
DDNS_STATUS_UNKNOWN	DDNS client has not yet been executed with this configuration.

Description

Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Files

Files

Name	Description
ddns.h	The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service.
ddns_config.h	Dynamic DNS configuration file

Description

This section lists the source and header files used by the library.

ddns.h

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service.

Enumerations

	Name	Description
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a preconfigured Dynamic DNS service.
	TCPIP_DDNS_Task	Standard TCP/IP stack module task function.
	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update.

Structures

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client

Description

DynDNS Headers for Microchip TCP/IP Stack API Header File

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

File Name

ddns.h

Company

Microchip Technology Inc.

ddns_config.h

Dynamic DNS configuration file

Macros

	Name	Description
	TCPIP_DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	TCPIP_DDNS_TASK_TICK_RATE	dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Dynamic Domain Name Service (DynDNS) Configuration file

This file contains the Dynamic DNS module configuration options

File Name

ddns_config.h

Company

Microchip Technology Inc.

FTP Module

This section describes the TCP/IP Stack Library FTP module.

Introduction

TCP/IP Stack Library File Transfer Protocol (FTP) Module for Microchip Microcontrollers

This library provides the API of the FTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

An embedded File Transfer Protocol (FTP) server is an excellent addition to any network-enabled device. FTP server capability facilitates the uploading of files to, and downloading of files from, an embedded device. Almost all computers have, at the very least, a command line FTP client that will allow a user to connect to an embedded FTP server.

Using the Library

This topic describes the basic architecture of the FTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ftp.h`

The interface to the FTP TCP/IP Stack library is defined in the `ftp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the FTP TCP/IP Stack library should include `tcpip.h`.

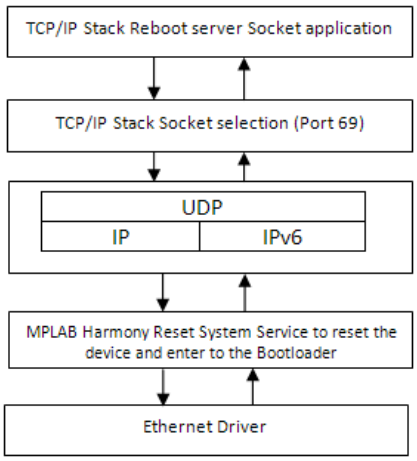
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the FTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

FTP Server Dependencies



Since the FTP server requires the use of the TCP/IP stack and a file system, which can be a MPFS or MDD, it also inherits the program memory and RAM requirements of those as well.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the FTP module.

Library Interface Section	Description
General Functions	This section provides a routine that performs FTP module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name
	TCPIP_FTPS_TASK_TICK_RATE	The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_FTP_TIMEOUT	FTP timeout, seconds

Description

The configuration of the FTP TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the FTP TCP/IP Stack. Based on the selections made, the FTP TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the FTP TCP/IP Stack.

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.

TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_FTP_MAX_CONNECTIONS Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_MAX_CONNECTIONS (1)
```

Description

Maximum number of FTP connections allowed

TCPIP_FTP_PASSWD_LEN Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_PASSWD_LEN (10)
```

Description

Specifies the max length of FTP login password

TCPIP_FTP_PUT_ENABLED Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_PUT_ENABLED
```

Description

Comment this line out to disable MPFS

TCPIP_FTP_USER_NAME_LEN Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_USER_NAME_LEN (10)
```

Description

Specifies the max length for user name

TCPIP_FTPS_TASK_TICK_RATE Macro

File

ftp_config.h

C

```
#define TCPIP_FTPS_TASK_TICK_RATE 33
```

Description

The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_FTP_TIMEOUT Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_TIMEOUT (1200ul)
```

Description

FTP timeout, seconds

Building the Library

This section lists the files that are available in the FTP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/ftp.c`	FTP 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 FTP module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) General Functions

	Name	Description
	TCPIP_FTP_ServerTask	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_FTP_MODULE_CONFIG	FTP Sever module runtime and initialization configuration data.

Description

This section describes the Application Programming Interface (API) functions of the FTP module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_FTP_ServerTask Function

Standard TCP/IP stack module task function.

File

ftp.h

C

```
void TCPIP_FTP_ServerTask();
```

Returns

None.

Description

This function performs FTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The FTP module should have been initialized.

Function

void TCPIP_FTP_ServerTask(void)

b) Data Types and Constants

TCPIP_FTP_MODULE_CONFIG Structure

FTP Sever module runtime and initialization configuration data.

File

ftp.h

C

```
typedef struct {
  uint16_t nConnections;
  uint16_t dataSktTxBuffSize;
  uint16_t dataSktRxBuffSize;
  char * userName;
  char * password;
} TCPIP_FTP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous FTP connections allowed
uint16_t dataSktTxBuffSize;	size of Data socket TX buffer for the associated socket; leave 0 for default
uint16_t dataSktRxBuffSize;	size of Data Socket RX buffer for the associated socket; leave 0 for default
char * userName;	FTP login User name. Size should not exceed more than TCPIP_FTP_USER_NAME_LEN
char * password;	FTP login password. Size should not exceed more than TCPIP_FTP_PASSWD_LEN

Description

Structure: TCPIP_FTP_MODULE_CONFIG

FTP server configuration and initialization data . Configuration is part of tcpip_stack_init.c.

Files

Files

Name	Description
ftp.h	An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device.
ftp_config.h	FTP configuration file

Description

This section lists the source and header files used by the library.

ftp.h

An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device.

Functions

	Name	Description
	TCPIP_FTP_ServerTask	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_FTP_MODULE_CONFIG	FTP Sever module runtime and initialization configuration data.

Description

FTP Server Definitions for the Microchip TCP/IP Stack

An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device. FTP server capability facilitates the uploading of files to, and downloading of files from, an embedded device. Almost all computers have, at the very least, a command line FTP client that will allow a user to connect to an embedded FTP server.

File Name

ftp.h

Company

Microchip Technology Inc.

ftp_config.h

FTP configuration file

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_TIMEOUT	FTP timeout, seconds
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name
	TCPIP_FTPS_TASK_TICK_RATE	The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

File Transfer Protocol (FTP) Configuration file

This file contains the FTP module configuration options

File Name

ftp_config.h

Company

Microchip Technology Inc.

HTTP Module

This section describes the TCP/IP Stack Library HTTP module.

Introduction

TCP/IP Stack Library Hypertext Transfer Protocol (HTTP) Module for Microchip Microcontrollers

This library provides the API of the HTTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The HTTP Web Server module allows a development board to act as a Web server. This facilitates an easy method to view status information and control applications using any standard Web browser.

Using the Library

This topic describes the basic architecture of the HTTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `http.h`

The interface to the HTTP TCP/IP Stack library is defined in the `http.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the HTTP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

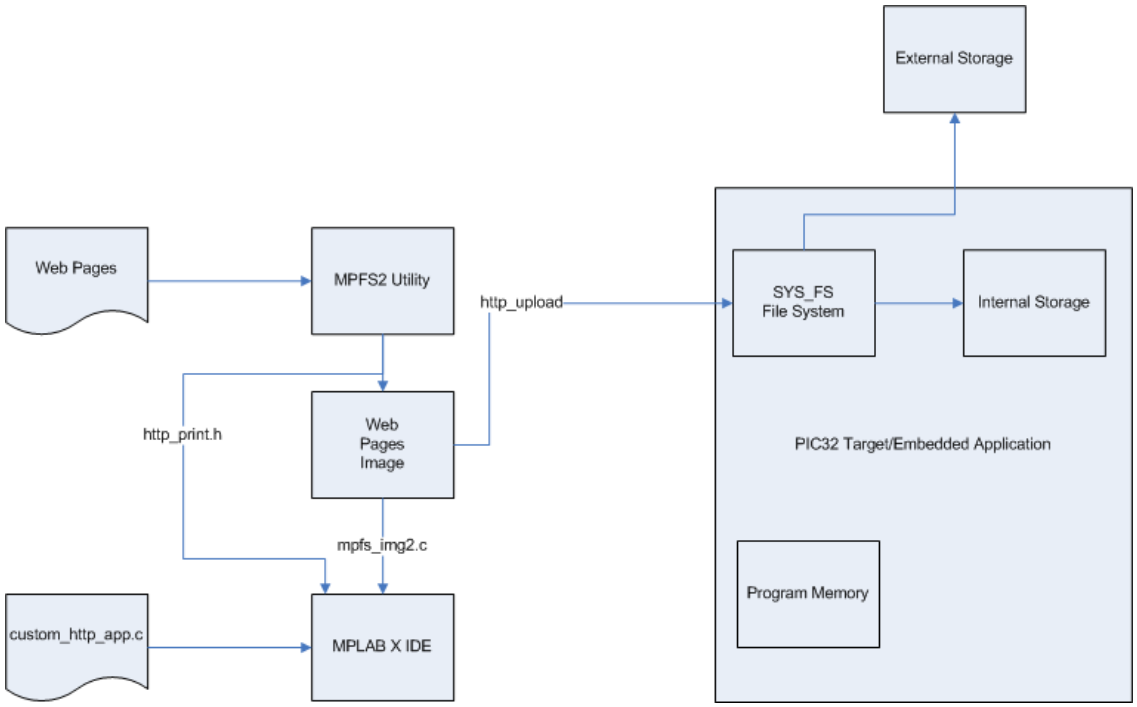
Abstraction Model

This library provides the API of the HTTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

HTTP Software Abstraction Block Diagram

Three main components are necessary to understand how the HTTP web server works: the web pages, the MPFS2 Utility, and the source files `custom_http_app.c` and `http_print.h`. An overview of the entire process is shown in the following diagram.



Web Pages

This includes all the HTML and associated images, CSS style sheets, and JavaScript files necessary to display the website. A sample application

including all these components is located in the `web_pages2` folder.

MPFS2 Utility

This program, supplied by Microchip, packages the web pages into a format that can be efficiently stored in either external non-volatile storage, or internal Flash program memory. This program also indexes dynamic variables found in the web pages and updates `http_print.h` with these indices. The MPFS2 Utility generates a SYS_FS compatible file image that can be directly uploaded on the PIC32 development board or stored in the Flash program memory as a source file image to be included in the project.

When dynamic variables are added or removed from your application, the MPFS2 Utility will update `http_print.h`. When this happens, the project must be recompiled in the MPLAB X IDE to ensure that all the new variable indices get added into the application.

custom_http_app.c

This file implements the Web application. It describes the output for dynamic variables (via TCPIP_HTTP_Print_varname callbacks), parses data submitted through forms (in TCPIP_HTTP_GetExecute and TCPIP_HTTP_PostExecutePost) and validates authorization credentials (in TCPIP_HTTP_FileAuthenticate and TCPIP_HTTP_UserAuthenticate).

The functionality of these callbacks is described within the demonstration application's web pages, and is also documented within the `custom_http_app.c` example that is distributed with the stack.

http_print.h

This file is generated automatically by the MPFS2 Utility. It indexes all of the dynamic variables and provides the "glue" between the variables located in the Web pages and their associated TCPIP_HTTP_Print_varname callback functions defined in `custom_http_app.c`. This file does not require modification by the programmer.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the HTTP module.

Library Interface Section	Description
Functions	Routines for Configuring the HTTP module
Data Types and Constants	This section provides various definitions describing this API

HTTP Features

The HTTP Web Server module has many capabilities. The following topics will introduce these features and provide examples.

Dynamic Variables

One of the most basic needs is to provide status information back to the user of your Web application. The HTTP server provides for this using dynamic variable substitution callbacks. These commands in your HTML code will alert the server to execute a callback function at that point, which the developer creates to write data into the web page. Dynamic Variables should be considered the output of your application.

Description

Basic Use

To create a dynamic variable, simply enclose the name of the variable inside a pair of tilde (~) characters within the web pages' HTML source code. (ex: ~myVariable~) When you run the MPFS2 Utility to generate the web pages, it will automatically index these variables in `http_print.h`. This index will instruct your application to invoke the function TCPIP_HTTP_Print_myVariable when this string is encountered.

Here is an example of using a dynamic variable to insert the build date of your application into the web pages:

```
<div class="examplebox code">~builddate~</div>
The associated callback will print the value into the web page:
void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
{
    TCP_SOCKET sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
}
```

Passing Parameters

You can also pass parameters to dynamic variables by placing numeric values inside of parenthesis after the variable name.

For example, ~led(2)~ will print the value of the second LED. The numeric values are passed as 16 bit values to your callback function. You can pass as many parameters as you wish to these functions, and if your C code has constants defined, those will be parsed as well. (ex: ~pair(3,TRUE)~).

The following code inserts the value of the push buttons into the web page, all using the same callback function:

```
<div class="examplebox code">btn(3)~ btn(2)~ btn(1)~ btn(0)~</div>
This associated callback will print the value of the requested button to the web page:
void TCPIP_HTTP_Print_btn(HTTP_CONN_HANDLE connHandle, uint16_t num)
{
    // Determine which button
    switch(num)
    {
        case 0:
            num = BUTTON0_IO;
            break;
        case 1:
            num = BUTTON1_IO;
            break;
        case 2:
            num = BUTTON2_IO;
            break;
        default:
            num = 0;
    }

    // Print the output
    TCPIP_TCP_StringPut(TCPIP_HTTP_CurrentConnectionSocketGet(connHandle), (num ? "up" : "down"));
}
```

Longer Outputs

The HTTP protocol operates in a fixed memory buffer for transmission, so not all data can be sent at once. Care must be taken inside of your callback function to avoid overrunning this buffer.

The callback functions must check how much space is available, write up to that many bytes, and then return. The callback will be invoked again when more space is free.

Note: For increased throughput performance, it is recommended that the callback should always try to write as much data as possible in each call.

To manage the output state, callbacks should make use of the callbackPos variable that is maintained by the Web server for each individual connection. This 32-bit value can be manipulated with the TCPIP_HTTP_CurrentConnectionCallbackPosGet and TCPIP_HTTP_CurrentConnectionCallbackPosSet functions.

The callbackPos is set to zero when a callback is first invoked. If a callback is only writing part of its output, it should set this field to a non-zero value to indicate that it should be called again when more space is available. This value will be available to the callback during the next call, which allows the function to resume output where it left off. A common use is to store the number of bytes written, or remaining to be written, in this field. Once the callback is finished writing its output, it must set callbackPos back to zero to indicate completion.

The following code outputs the value of the build date of the application into the web pages:

```
<div class="examplebox code">~builddate~</div>

void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
{
    TCP_SOCKET sktHTTP;
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x01);
    if(TCPIP_TCP_PutIsReady(sktHTTP) < strlen((const char*)__DATE__" "__TIME__))
        return;

    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x00);
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
}
```

The initial call to TCPIP_TCP_PutIsReady determines how many bytes can be written to the buffer right now. When there is not enough buffer space for performing the output operation the function sets the current value of the callbackPos and returns. Once the output operation is performed the callbackPos is cleared to '0'.

Including Files

Often it is useful to include the entire contents of another file in your output. Most web pages have at least some portion that does not change, such as the header, menu of links, and footer. These sections can be abstracted out into separate files which makes them easier to manage and conserves storage space.

To include the entire contents of another file, use a dynamic variable that starts with "inc:", such as ~inc:header.inc~.

This sequence will cause the file header.inc to be read from the file system and inserted at this location.

The following example indicates how to include a standard menu bar section into every page:

```
<div id="menu">~inc:menu.inc~</div>
```

At this time, dynamic variables are not recursive, so any variables located inside files included in this manner are not parsed.

Form Processing

Many applications need to accept data from a user. A common solution is to present a form to the user in a Web page, and then have the device process the values submitted via this form. Web forms are usually submitted using one of two methods (GET and POST), and the HTTP Web server supports both.

Description

The GET Method

The GET method appends the data to the end of the URI. This data follows the question mark (?) in the browser's address bar. (ex: http://mchpboard/form.htm?led1=0&led2=1&led3=0) Data sent via GET is automatically decoded and stored in the current connection data buffer. Since it is to be stored in memory, this data is limited to the size of the connection data buffer which by default is 100 bytes (configurable by HTTP_MAX_DATA_LEN build time symbol or by the run time data passes at the HTTP module initialization).

It is generally easier to process data received in this manner.

The callback function TCPIP_HTTP_GetExecute is implemented by the application developer to process this data and perform any necessary actions. The function TCPIP_HTTP_ArgGet provides an easy method to retrieve submitted values for processing.

See the custom _http_app.c for an example of TCPIP_HTTP_GetExecute implementation.

The POST Method

The POST method transmits data after all the request headers have been sent. This data is not visible in the browser's address bar, and can only be seen with a packet capture tool. It does however use the same URL encoding method.

The HTTP server does not perform any preparsing of this data. All POST data is left in the TCP buffer, so the custom application will need to access the TCP buffer directly to retrieve and decode it. The functions TCPIP_HTTP_PostNameRead and TCPIP_HTTP_PostValueRead have been provided to assist with these requirements. However, these functions can only be used when at least entire variables are expected to fit in the TCP buffer at once. Most POST processing functions will be implemented as state machines to use these functions. There is a status variable per each connection that stores the current state accessible with TCPIP_HTTP_CurrentConnectionPostSmGet and TCPIP_HTTP_CurrentConnectionPostSmSet functions. This state machine variable is reset to zero with each new request. Functions should generally implement a state to read a variable name, and another to read an expected value. Additional states may be helpful depending on the application.

The following example form accepts an e-mail address, a subject, and a message body. Since this data will likely total over the size of the internal connection data buffer, it should be submitted via POST.

```
<form method="post" action="/email.htm">
To: <input type="text" name="to" maxlength="50" /><br />
Subject: <input type="text" name="subject" maxlength="50" /><br />
Message:<br />
<textarea name="msg" rows="6"></textarea><br />
<input type="submit" value="Send Message" /></div>
</form>
```

Suppose a user enters the following data into this form:

To: joe@picsaregood.com

Subject: Sent by a PIC

Message: I sent this message using my development board!

The TCPIP_HTTP_PostExecute function will be called with the following data still in the TCP buffer:

to=joe%40picsaregood.com&subject=Sent+by+a+PIC&msg=I+sent+this+message+using+my+development+board%21

To use the e-mail module, the application needs to read in the address and the subject, store those in RAM, and then send the message. However, since the message is not guaranteed to fit in RAM all at once, it must be read as space is available and passed to the e-mail module. A state machine, coupled with the TCPIP_HTTP_PostNameRead and

TCPIP_HTTP_PostValueRead functions can simplify this greatly.

See the TCPIP_HTTP_PostExecute and HTTPPostEmail functions in the supplied custom_http_app.c for a complete implementation of this example.

Authentication

The HTTP protocol provides a method for servers to request a user name and password from a client before granting access to a page. The HTTP server supports this authentication mechanism, allowing developers to require valid credentials for access to some or all pages.

Description

Authentication functionality is supported by two user-provided callback functions. The first, TCPIP_HTTP_FileAuthenticate, determines if the requested page requires valid credentials to proceed. The second, TCPIP_HTTP_UserAuthenticate, checks the user name and password against

an accepted list and determines whether to grant or deny access. The connection stores the current authorization setting which can be manipulated by using the TCPIP_HTTP_CurrentConnectionIsAuthorizedGet and TCPIP_HTTP_CurrentConnectionIsAuthorizedSet functions.

Requiring Authentication

When a request is first made, the function TCPIP_HTTP_FileAuthenticate is called to determine if that page needs password protection. This function returns a value to instruct the HTTP server how to proceed. The most significant bit indicates whether or not access is granted. That is, values 0x80 and higher allow access unconditionally, while values 0x79 and lower will require a user name and password at a later point. The value returned is stored in the connection data so that it can be accessed by future callback functions.

The following example is the simplest case, in which all files require a password for access:

```
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)
{
return 0;
}
In some cases, only certain files will need to be protected. The second example requires a password for any
file located in the /treasure folder:
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)
{
    // Compare to "/treasure" folder. Don't use strcmp here, because
    // cFile has additional path info such as "/treasure/gold.htm"
    if(memcmp(cFile, (const void*)"treasure", 8) == 0)
    {   // Authentication will be needed later
        return 0;
    }

    // No authentication required
    return 0x80;
}
```

More complex uses could require an administrative user to access the `/admin` folder, while any authenticated user can access the rest of the site. This requires the TCPIP_HTTP_FileAuthenticate to return different authentication values for a different file.

Validating Credentials

The TCPIP_HTTP_UserAuthenticate function determines if the supplied user name and password are valid to access the specific resource. Again, the most significant bit indicates whether or not access is granted. The value returned is also stored in the connection internal data and it can be accessed by future callback functions.

The following example is the simplest case, in which one user/password pair is accepted for all pages:

```
uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass)
{
    if(!strcmp((char*)cUser, (const char*)"AliBaba") &&
    !strcmp((char*)cPass, (const char*)"Open Sesame!") )
    {
        return 0x80;
    }

    return 0;
}
```

More complex uses are certainly feasible. Many applications may choose to store the user names and passwords in protected files so that they may be updated by a privileged user. In some cases, you may have multiple users with various levels of access. The application may wish to return various values above 0x80 in TCPIP_HTTP_UserAuthenticate so that later callback functions can determine which user logged in.

Cookies

By design, HTTP is a session-less and state-less protocol; every connection is an independent session with no relation to another.

Description

Cookies were added to the protocol description to solve this problem. This feature allows a web server to store small bits of text in a user's browser. These values will be returned to the server with every request, allowing the server to associate session variables with a request. Cookies are typically used for more advanced authentication systems.

Best practice is generally to store the bulk of the data on the server, and store only a unique identifier with the browser. This cuts down on data overhead and ensures that the user cannot modify the values stored with the session. However, logic must be implemented in the server to expire old sessions and allocate memory for new ones. If sensitive data is being stored, it is also important that the identifier be random enough so as to prevent stealing or spoofing another user's cookies.

Retrieving Cookies

In the HTTP server, cookies are retrieved automatically. They are stored in the connection internal data buffer just as any other GET form argument or URL parameter would be. The proper place to parse these values is therefore in the TCPIP_HTTP_GetExecute callback using the

TCPIP_HTTP_ArgGet.

This model consumes some of the limited space available for URL parameters. Ensure that cookies do not consume more space than is available (as defined by HTTP_MAX_DATA_LEN) and that they will fit after any data that may be submitted via a GET form. If enough space is not available, the cookies will be truncated.

Setting Cookies

Cookies can be set in TCPIP_HTTP_GetExecute or TCPIP_HTTP_PostExecute. To set a cookie, store the name/value pairs in the connection internal buffer data as a series of null-terminated strings. Then, call TCPIP_HTTP_CurrentConnectionHasArgsSet with a parameter equal to the number of name/value pairs to be set. For example, the following code sets a cookie indicating a user's preference for a type of cookie:

```
void TCPIP_HTTP_GetExecute(void)
{
...
// Set a cookie
uint8_t* connData = TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle);
uint16_t httpBuffSize  = TCPIP_HTTP_CurrentConnectionDataBufferSizeGet(connHandle);
if(strlen(“flavor”) + strlen(“oatmeal raisin) + 2 <= httpBuffSize)
{
  strcpy((char*)connData, "flavor");
  strcpy((char*)connData + strlen("flavor") + 1, "oatmeal raisin");
  TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle, 1);
}

...
}
```

After this, all future requests from this browser will include the parameter "flavor" in the connection data along with the associated value of "oatmeal raisin".

Compression

All modern web browsers can receive files encoded with GZIP compression. For static files (those without dynamic variables), this can decrease the amount of data transmitted by as much as 60%.

Description

The MPFS2 Utility will automatically determine which files can benefit from GZIP compression, and will store the compressed file in the MPFS2 image when possible. This generally includes all JavaScript and CSS files. (Images are typically already compressed, so the MPFS2 Utility will generally decide it is better to store them uncompressed.) This HTTP server will then seamlessly return this compressed file to the browser. Less non-volatile storage space will be required for storing the pages image, and faster transfers back to the client will result. No special configuration is required for this feature.

To prevent certain extensions from being compressed, use the Advanced Settings dialog in the MPFS2 Utility.

Configuring the Library

Macros

	Name	Description
	TCPIP_STACK_USE_BASE64_DECODE	Authentication requires Base64 decoding Enable basic authentication support
	TCPIP_HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	TCPIP_HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_MAX_HEADER_LEN	Set to length of longest string above
	TCPIP_HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	TCPIP_HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

	TCPIP_HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTPS_DEFAULT_FILE	Indicate what HTTPS file to serve when no specific one is requested
	TCPIP_HTTP_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.
	TCPIP_HTTP_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NO_AUTH_WITHOUT_SSL	Uncomment to require secure connection before requesting a password

Description

The configuration of the HTTP TCP/IP Stack is based on the file `tcpip_config.h` (which may include `http_config.h`).

This header file contains the configuration selection for the HTTP TCP/IP Stack. Based on the selections made, the HTTP TCP/IP Stack 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.

TCPIP_STACK_USE_BASE64_DECODE Macro

File

http_config.h

C

```
#define TCPIP_STACK_USE_BASE64_DECODE
```

Description

Authentication requires Base64 decoding Enable basic authentication support

TCPIP_HTTP_CACHE_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_CACHE_LEN ("600")
```

Description

Max lifetime (sec) of static responses as string

TCPIP_HTTP_CONFIG_FLAGS Macro

File

http_config.h

C

```
#define TCPIP_HTTP_CONFIG_FLAGS 0
```

Description

Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values

TCPIP_HTTP_DEFAULT_FILE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTP file to serve when no specific one is requested

TCPIP_HTTP_DEFAULT_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_DEFAULT_LEN (10u)
```

Description

For buffer overrun protection. Set to longest length of above two strings.

TCPIP_HTTP_MAX_CONNECTIONS Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_CONNECTIONS (4)
```

Description

Maximum numbers of simultaneous supported HTTP connections.

TCPIP_HTTP_MAX_DATA_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_DATA_LEN (100u)
```

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

TCPIP_HTTP_MAX_HEADER_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_HEADER_LEN (15u)
```

Description

Set to length of longest string above

TCPIP_HTTP_MIN_CALLBACK_FREE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MIN_CALLBACK_FREE (16u)
```

Description

Define the minimum number of bytes free in the TX FIFO before executing callbacks

TCPIP_HTTP_SKT_RX_BUFF_SIZE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_SKT_TX_BUFF_SIZE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_TIMEOUT Macro

File

http_config.h

C

```
#define TCPIP_HTTP_TIMEOUT (45u)
```

Description

Max time (sec) to await more data before timing out and disconnecting the socket

TCPIP_HTTP_USE_AUTHENTICATION Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_AUTHENTICATION
```

Description

Enable basic authentication support

TCPIP_HTTP_USE_COOKIES Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_COOKIES
```

Description

Enable cookie support

TCPIP_HTTP_USE_POST Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_POST
```

Description

Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

TCPIP_HTTPS_DEFAULT_FILE Macro

File

http_config.h

C

```
#define TCPIP_HTTPS_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTPS file to serve when no specific one is requested

TCPIP_HTTP_FILE_UPLOAD_ENABLE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_FILE_UPLOAD_ENABLE
```

Description

Configure MPFS over HTTP updating Comment this line to disable updating via HTTP

TCPIP_HTTP_FILE_UPLOAD_NAME Macro

File

http_config.h

C

```
#define TCPIP_HTTP_FILE_UPLOAD_NAME "mpfsupload"
```

Description

This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.

TCPIP_HTTP_TASK_RATE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_TASK_RATE 33
```

Description

The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_HTTP_NO_AUTH_WITHOUT_SSL Macro

File

http_config.h

C

```
#define TCPIP_HTTP_NO_AUTH_WITHOUT_SSL
```

Description

Uncomment to require secure connection before requesting a password

Building the Library

This section lists the files that are available in the HTTP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/http.c`	HTTP Server 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 HTTP module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) Functions

	Name	Description
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.

	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_FileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_GetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_PostExecute	Processes POST form variables and data.
	TCPIP_HTTP_Print_varname	Inserts dynamic content into a web page
	TCPIP_HTTP_UserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket.
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.
	TCPIP_HTTP_PostReadPair	Reads a name and value pair from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_CurrentConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_CurrentConnectionIndexGet	Gets the index of the HTTP connection.

b) Data Types and Constants

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	HTTP_FILE_TYPE	File type definitions
	HTTP_IO_RESULT	Result states for execution callbacks
	HTTP_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	HTTP_READ_STATUS	Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair
	HTTP_STATUS	Supported Commands and Server Response Codes
	TCPIP_HTTP_MODULE_CONFIG	HTTP module dynamic configuration data

Description

This section describes the Application Programming Interface (API) functions of the HTTP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_HTTP_ArgGet Function

Locates a form field value in a given data array.

File

http.h

C

```
const uint8_t* TCPIP_HTTP_ArgGet(const uint8_t* cData, const uint8_t* cArg);
```

Returns

A pointer to the argument value, or NULL if not found.

Description

This function searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

Remarks

None.

Preconditions

The data array has a valid series of null terminated name/value pairs.

Example

```
void TCPIP_HTTP_Print_cookiename(HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    TCP_SOCKET sktHTTP;

    ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    if(ptr)
        TCPIP_TCP_StringPut(sktHTTP, ptr);
    else
        TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
}
```

Parameters

Parameters	Description
cData	the buffer to search
cArg	the name of the argument to find

Function

const uint8_t* TCPIP_HTTP_ArgGet(const uint8_t* cData, const uint8_t* cArg)

TCPIP_HTTP_CurrentConnectionByteCountDec Function

Decrements the connection byte count.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionByteCountDec(HTTP_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function decrements the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count reduction

Function

void TCPIP_HTTP_CurrentConnectionByteCountDec(HTTP_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_CurrentConnectionByteCountGet Function

Returns how many bytes have been read so far

File

http.h

C

```
uint32_t TCPIP_HTTP_CurrentConnectionByteCountGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Current connection byte count, how many bytes have been read so far.

Description

This function returns the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Example

```
switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    case SM_CFG_SNMP_READ_NAME:
        // If all parameters have been read, end
        if(TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle) == 0u)
        {
            return HTTP_IO_DONE;
        }
      .
      .
      .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_CurrentConnectionByteCountGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionByteCountSet Function

Sets how many bytes have been read so far.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionByteCountSet(HTTP_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function sets the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count to be set

Function

void TCPIP_HTTP_CurrentConnectionByteCountSet(HTTP_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_CurrentConnectionCallbackPosGet Function

Returns the callback position indicator.

File

http.h

C

```
uint32_t TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Callback position indicator for connection defined by connHandle.

Description

This function will return the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables.

Remarks

None.

Preconditions

None.

Example

```
  uint32_t callbackPos;
  callbackPos = TCPIP_HTTP_CurrentConnectionCallbackPosGet(connHandle);
  if(callbackPos == 0x00u)
      callbackPos = (uint32_t)DDNSClient.Host.szRAM;
  callbackPos = (uint32_t)TCPIP_TCP_StringPut(TCPIP_HTTP_CurrentConnectionSocketGet(connHandle),
(uint8_t*)callbackPos);
  if(*(uint8_t*)callbackPos == '0')
      callbackPos = 0x00;
  TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, callbackPos);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionCallbackPosSet Function

Sets the callback position indicator.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionCallbackPosSet(HTTP_CONN_HANDLE connHandle, uint32_t callbackPos);
```

Returns

None.

Description

This function will set the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables. When set to a value != 0, it indicates to the HTTP server that the application has more pending processing that needs to be done.

Remarks

None.

Preconditions

None.

Example

```
void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
{
    TCP_SOCKET sktHTTP;
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x01);
    if(TCPIP_TCP_PutIsReady(sktHTTP) < strlen((const char*)__DATE__" "__TIME__))
    { // Don't have room to output build date and time
        return;
    }
    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x00);
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
callbackPos	new connection callback position value

Function

void TCPIP_HTTP_CurrentConnectionCallbackPosSet(HTTP_CONN_HANDLE connHandle, uint32_t callbackPos)

TCPIP_HTTP_CurrentConnectionHasArgsGet Function

Checks whether there are get or cookie arguments.

File

http.h

C

```
uint8_t TCPIP_HTTP_CurrentConnectionHasArgsGet(HTTP_CONN_HANDLE connHandle);
```

Returns

The current value of the connection hasArgs.

Description

The function will get the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
uint8_t hasArgs = TCPIP_HTTP_CurrentConnectionHasArgsGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_CurrentConnectionHasArgsGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionHasArgsSet Function

Sets whether there are get or cookie arguments.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionHasArgsSet(HTTP_CONN_HANDLE connHandle, uint8_t args);
```

Returns

None.

Description

The function sets the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
else if(!memcmp(filename, "cookies.htm", 11))
{
    TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle, true);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
args	boolean if there are arguments or not

Function

void TCPIP_HTTP_CurrentConnectionHasArgsSet(HTTP_CONN_HANDLE connHandle, uint8_t args)

TCPIP_HTTP_CurrentConnectionPostSmSet Function

Set the POST state machine state.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionPostSmSet(HTTP_CONN_HANDLE connHandle, uint16_t state);
```

Returns

None.

Description

This function sets the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
uint8_t* httpDataBuff;
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:

        // Read a name
        if(TCPIP_HTTP_PostNameRead(connHandle, httpDataBuff, HTTP_MAX_DATA_LEN) == HTTP_READ_INCOMPLETE)
            return HTTP_IO_NEED_DATA;

        TCPIP_HTTP_CurrentConnectionPostSmSet(connHandle, SM_POST_LCD_READ_VALUE);
        // No break...continue reading value

    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
     .
     .
     .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
state	16 bit integer state for POST state machine

Function

void TCPIP_HTTP_CurrentConnectionPostSmSet(HTTP_CONN_HANDLE connHandle, uint16_t state)

TCPIP_HTTP_CurrentConnectionStatusGet Function

Gets HTTP status.

File

http.h

C

```
HTTP_STATUS TCPIP_HTTP_CurrentConnectionStatusGet(HTTP_CONN_HANDLE connHandle);
```

Returns

A HTTP_STATUS value.

Description

This function returns the current HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
HTTP_STATUS currStat =  TCPIP_HTTP_CurrentConnectionStatusGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_STATUS TCPIP_HTTP_CurrentConnectionStatusGet(HTTP_CONN_HANDLE connHandle);

TCPIP_HTTP_FileAuthenticate Function

Determines if a given file name requires authentication

File

http.h

C

```
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile);
```

Returns

- <= 0x79 - valid authentication is required
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved for the current connection and can be read using TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(). It will be available to future callbacks, including TCPIP_HTTP_UserAuthenticate and any of the TCPIP_HTTP_GetExecute, TCPIP_HTTP_PostExecute, or TCPIP_HTTP_Print_varname callbacks.

Remarks

This function may NOT write to the TCP buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file being requested

Function

uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)

TCPIP_HTTP_GetExecute Function

Processes GET form field variables and cookies.

File

http.h

C

```
HTTP_IO_RESULT TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
```

Returns

- HTTP_IO_DONE - application is done processing
- HTTP_IO_NEED_DATA - this value may not be returned because more data will not become available
- HTTP_IO_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, the connection data buffer (see TCPIP_HTTP_CurrentConnectionDataBufferGet()) contains sequential name/value pairs of strings representing the data received. In this format, TCPIP_HTTP_ArgGet can be used to search for specific variables in the input. If data buffer space associated with this connection is required, connection data buffer may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_PostExecute and any TCPIP_HTTP_Print_varname dynamic substitutions.

This function may also issue redirections by setting the connection data buffer to the destination file name or URL, and the connection httpStatus (TCPIP_HTTP_CurrentConnectionStatusSet()) to HTTP_REDIRECT.

Finally, this function may set cookies. Set connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive) and then set the connection hasArgs (TCPIP_HTTP_CurrentConnectionHasArgsSet()) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments. This function may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet()) or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_IO_RESULT TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_PostExecute Function

Processes POST form variables and data.

File

http.h

C

```
HTTP_IO_RESULT TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);
```

Returns

- HTTP_IO_DONE - application is done processing
- HTTP_IO_NEED_DATA - more data is needed to continue, and this function should be called again later
- HTTP_IO_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the TCP buffer. The connection byteCount (see TCPIP_HTTP_CurrentConnectionByteCountGet will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the TCP buffer, the application must call TCPIP_TCP_ArrayGet in order to retrieve bytes. When this is done, connection byteCount MUST be updated to reflect how many bytes now remain. The functions TCPIP_TCP_ArrayFind and TCPIP_TCP_Find may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The TCPIP_HTTP_URLDecode function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, the connection data buffer (see TCPIP_HTTP_CurrentConnectionDataBufferGet()) may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_PostExecute and any TCPIP_HTTP_Print_varname dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting the connection data buffer to the destination file or URL, and the connection httpStatus (TCPIP_HTTP_CurrentConnectionStatusSet()) to HTTP_REDIRECT.

Finally, this function may set cookies. Set the connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive), and then set the connection hasArgs (TCPIP_HTTP_CurrentConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called when the request method is POST, and is only used when HTTP_USE_POST is defined. This method may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet) or connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_IO_RESULT TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_Print_varname Function

Inserts dynamic content into a web page

File

http.h

C

```
void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1, uint16_t wParam2, ...);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. These functions generate dynamic content to be inserted into web pages and other files returned by the HTTP server.

Functions of this type are called when a dynamic variable is located in a web page. (i.e., ~varname~) The name between the tilde '~' characters is appended to the base function name. In this example, the callback would be named TCPIP_HTTP_Print_varname.

The function prototype is located in your project's http_print.h, which is automatically generated by the mpfs2.jar utility. The prototype will have uint16_t parameters included for each parameter passed in the dynamic variable. For example, the variable "~myArray(2,6)~" will generate the prototype "void TCPIP_HTTP_Print_varname(uint16_t, uint16_t);".

When called, this function should write its output directly to the TCP socket using any combination of TCPIP_TCP_PutIsReady, TCPIP_TCP_Put, TCPIP_TCP_ArrayPut, TCPIP_TCP_StringPut, TCPIP_TCP_ArrayPut, and TCPIP_TCP_StringPut.

Before calling, the HTTP server guarantees that at least HTTP_MIN_CALLBACK_FREE bytes (defaults to 16 bytes) are free in the output buffer. If the function is writing less than this amount, it should simply write the data to the socket and return.

In situations where a function needs to write more this amount, it must manage its output state using the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet/TCPIP_HTTP_CurrentConnectionCallbackPosSet). This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. Typically this is used to track how many bytes have been written, or how many remain to be written. If the connection callbackPos is non-zero, the function will be called again when more buffer space is available. Once the callback completes, set this value back to zero to resume normal servicing of the request.

Remarks

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global

or static variables inside this routine. Use the connection callbackPos or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
wParam1	first parameter passed in the dynamic variable (if any)
wParam2	second parameter passed in the dynamic variable (if any)
...	additional parameters as necessary

Function

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle)

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1)

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1, uint16_t wParam2, ...)

TCPIP_HTTP_UserAuthenticate Function

Performs validation on a specific user name and password.

File

http.h

C

```
uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass);
```

Returns

- <= 0x79 - the credentials were rejected
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if the user name and password supplied by the client are acceptable for this resource. This callback function can thus to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The value returned by this function is stored in the corresponding connection data and will be available with TCPIP_HTTP_CurrentConnectionIsAuthorizedGet in any of the TCPIP_HTTP_GetExecute, TCPIP_HTTP_PostExecute, or TCPIP_HTTP_Print_varname callbacks.

Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the TCP buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cUser	the user name supplied by the client
cPass	the password supplied by the client

Function

uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass)

TCPIP_HTTP_CurrentConnectionDataBufferGet Function

Returns pointer to connection general purpose data buffer.

File

http.h

C

```
uint8_t* TCPIP_HTTP_CurrentConnectionDataBufferGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Pointer to the connection's general purpose data buffer.

Description

This function returns a pointer to the HTTP connection internal data buffer. This gives access to the application to the data that's stored in the HTTP connection buffer.

Remarks

None.

Preconditions

None.

Example

```
void TCPIP_HTTP_Print_cookiename(HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    TCP_SOCKET sktHTTP;

    ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    if(ptr)
        TCPIP_TCP_StringPut(sktHTTP, ptr);
    else
        TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t* TCPIP_HTTP_CurrentConnectionDataBufferGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionFileGet Function

Get handle to current connection's file.

File

http.h

C

```
SYS_FS_HANDLE TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Handle to File System file belonging to the connection defined by connHandle

Description

This function returns the handle of the current HTTP connection file.

Remarks

None.

Preconditions

None.

Example

```
uint8_t myBuff[20];

// Get the file handle and read from that file
SYS_FS_FileRead(myBuff, sizeof(myBuff), TCPIP_HTTP_CurrentConnectionFileGet(connHandle));
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

FILE_HANDLE TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionIsAuthorizedGet Function

Gets the authorized state for the current connection.

File

http.h

C

```
uint8_t TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(HTTP_CONN_HANDLE connHandle);
```

Returns

A uint8_t representing the authorization status.

Description

This function returns the authorization status for the current HTTP connection. This is one of the values returned by the TCPIP_HTTP_FileAuthenticate() function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t isAuth;

isAuth = TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionIsAuthorizedSet Function

Sets the authorized state for the current connection.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(HTTP_CONN_HANDLE connHandle, uint8_t auth);
```

Returns

None.

Description

This function sets the authorization status for the current HTTP connection. This has to be one of the values in the set returned by the TCPIP_HTTP_FileAuthenticate() function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t auth = 0x80;

TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(connHandle, auth);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
auth	new authorization state

Function

void TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(HTTP_CONN_HANDLE connHandle, uint8_t auth)

TCPIP_HTTP_CurrentConnectionPostSmGet Function

Get the POST state machine state.

File

http.h

C

```
uint16_t TCPIP_HTTP_CurrentConnectionPostSmGet(HTTP_CONN_HANDLE connHandle);
```

Returns

16-bit integer POST state machine state.

Description

This function returns the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:
     .
```

```
        .
        .
      // Found the value, so store the LCD and return
      case SM_POST_LCD_READ_VALUE:
        .
        .
        .
  }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_CurrentConnectionPostSmGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionSocketGet Function

Get the socket for the current connection.

File

http.h

C

```
TCP_SOCKET TCPIP_HTTP_CurrentConnectionSocketGet(HTTP_CONN_HANDLE connHandle);
```

Returns

TCP_SOCKET for the connection defined by connHandle.

Description

The function returns the TCP socket of the specified HTTP connection. The user gets access to the connection socket which it can use for sending/reading data.

Remarks

None.

Preconditions

None.

Example

```
uint32_t byteCount;
TCP_SOCKET sktHTTP;

byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{   // Configuration Failure
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
    return HTTP_IO_DONE;
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

TCP_SOCKET TCPIP_HTTP_CurrentConnectionSocketGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionStatusSet Function

Sets HTTP status.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionStatusSet(HTTP_CONN_HANDLE connHandle, HTTP_STATUS stat);
```

Returns

None.

Description

Allows write access to the HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{   // Configuration Failure
    // 302 Redirect will be returned
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
stat	new HTTP_STATUS enumeration value.

Function

void TCPIP_HTTP_CurrentConnectionStatusSet(HTTP_CONN_HANDLE connHandle, HTTP_STATUS stat)

TCPIP_HTTP_CurrentConnectionUserDataGet Function

Gets the user data parameter for the current connection.

File

http.h

C

```
const void* TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle);
```

Returns

User data that's stored as part of the connection.

Description

This function returns the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It can be set by the user with TCPIP_HTTP_CurrentConnectionUserDataSet().

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

myConnData = (uint32_t)TCPIP_HTTP_CurrentConnectionUserDataGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

const void* TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionUserDataSet Function

Sets the user data parameter for the current connection.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData);
```

Returns

None.

Description

This function will set the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It is available to the user by calling TCPIP_HTTP_CurrentConnectionUserDataGet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

TCPIP_HTTP_CurrentConnectionUserDataSet(connHandle, (const void*)myConnData);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
uData	user supplied data

Function

void TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData)

TCPIP_HTTP_FileInclude Function

Writes a file byte-for-byte to the currently loaded TCP socket.

File

http.h

C

```
void TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile);
```

Returns

None.

Description

This function allows an entire file to be included as a dynamic variable, providing a basic templating system for HTML web pages. This reduces unneeded duplication of visual elements such as headers, menus, etc.

When pHttpCon->callbackPos is 0, the file is opened and as many bytes as possible are written. The current position is then saved to pHttpCon->callbackPos and the file is closed. On subsequent calls, reading begins at the saved location and continues. Once the end of the input

file is reached, pHttpCon->callbackPos is set back to 0 to indicate completion.

Remarks

Users should not call this function directly, but should instead add dynamic variables in the form of `~inc:filename.ext~` in their HTML code to include (for example) the file "filename.ext" at that specified location. The mpfs2.jar utility will handle the rest.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file to be sent

Function

void TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile)

TCPIP_HTTP_PostNameRead Function

Reads a name from a URL encoded string in the TCP buffer.

File

http.h

C

```
HTTP_READ_STATUS TCPIP_HTTP_PostNameRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen);
```

Returns

- HTTP_READ_OK - name was successfully read
- HTTP_READ_TRUNCTATED - entire name could not fit in the buffer, so the value was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire name was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a name from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

Remarks

None.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostNameRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_PostValueRead Function

Reads a value from a URL encoded string in the TCP buffer.

File

http.h

C

```
HTTP_READ_STATUS TCPIP_HTTP_PostValueRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen);
```

Returns

- HTTP_READ_OK - value was successfully read
- HTTP_READ_TRUNCTATED - entire value could not fit in the buffer, so the value was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire value was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a value from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If pHttpCon->byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

Remarks

None.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the value once it is read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostValueRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_URLDecode Function

Parses a string from URL encoding to plain-text.

File

http.h

C

```
uint8_t* TCPIP_HTTP_URLDecode(uint8_t* cData);
```

Returns

A pointer to the last null terminator in data, which is also the first free byte for new data.

Description

This function parses a string from URL encoding to plain-text. The following conversions are made: ‘=’ to ‘0’, ‘&’ to ‘0’, ‘+’ to ‘ ‘, and “%xx” to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

Remarks

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

Parameters	Description
cData	The string which is to be decoded in place.

Function

uint8_t* TCPIP_HTTP_URLDecode(uint8_t* cData)

TCPIP_HTTP_PostReadPair Macro

Reads a name and value pair from a URL encoded string in the TCP buffer.

File

http.h

C

```
#define TCPIP_HTTP_PostReadPair(connHandle, cData, wLen) TCPIP_HTTP_PostValueRead(connHandle, cData, wLen)
```

Returns

- HTTP_READ_OK - name and value were successfully read
- HTTP_READ_TRUNCTATED - entire name and value could not fit in the buffer, so input was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire name and value was not yet in the buffer, so call this function again later to retrieve

Description

Reads a name and value pair from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates the connection byteCount (see TCPIP_HTTP_CurrentConnectionByteCountGet()) by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Remarks

This function is aliased to TCPIP_HTTP_PostValueRead, since they effectively perform the same task. The name is provided only for completeness.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostReadPair(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_Task Function

Standard TCP/IP stack module task function.

File

http.h

C

```
void TCPIP_HTTP_Task();
```

Returns

None.

Description

This function performs HTTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The HTTP module should have been initialized.

Function

void TCPIP_HTTP_Task(void)

TCPIP_HTTP_ActiveConnectionCountGet Function

Gets the number of active connections.

File

http.h

C

```
int TCPIP_HTTP_ActiveConnectionCountGet(int* pOpenCount);
```

Returns

The number of active and total connections.

Description

This function will return the number of active and total HTTP connections at the current time.

Remarks

The value returned by this function is informational only. The number of active connections changes dynamically.

Preconditions

None.

Example

```
int nConns;

nConns = TCPIP_HTTP_ActiveConnectionCountGet(0);
```

Parameters

Parameters	Description
pOpenCount	address to store the number of total opened connections Could be NULL if not needed.

Function

int TCPIP_HTTP_ActiveConnectionCountGet(int* pOpenCount);

TCPIP_HTTP_CurrentConnectionHandleGet Function

Gets the connection handle of a HTTP connection.

File

http.h

C

```
HTTP_CONN_HANDLE TCPIP_HTTP_CurrentConnectionHandleGet(int connIx);
```

Returns

A valid connection handle if the connection index is valid 0 if there is no such connection

Description

This function will return the connection handle of the requested HTTP connection index.

Remarks

None

Preconditions

None.

Example

```
HTTP_CONN_HANDLE connHandle;

connHandle = TCPIP_HTTP_CurrentConnectionHandleGet(0);
```

Parameters

Parameters	Description
connIx	the HTTP connection ix.

Function

HTTP_CONN_HANDLE TCPIP_HTTP_CurrentConnectionHandleGet(int connIx);

TCPIP_HTTP_CurrentConnectionIndexGet Function

Gets the index of the HTTP connection.

File

http.h

C

```
int TCPIP_HTTP_CurrentConnectionIndexGet(HTTP_CONN_HANDLE connHandle);
```

Returns

The connection index.

Description

This function will return the index of the requested HTTP connection.

Remarks

None

Preconditions

None.

Example

```
int connIx;

connIx = TCPIP_HTTP_CurrentConnectionIndexGet(connHandle);
```

Parameters

Parameters	Description
connHandle	the HTTP connection handle.

Function

int TCPIP_HTTP_CurrentConnectionIndexGet(HTTP_CONN_HANDLE connHandle);

b) Data Types and Constants

HTTP_CONN_HANDLE Type

File

http.h

C

```
typedef const void* HTTP_CONN_HANDLE;
```

Description

HTTP connection identifier, handle of a HTTP connection

HTTP_FILE_TYPE Enumeration

File

http.h

C

```
typedef enum {
  HTTP_TXT = 0u,
  HTTP_HTM,
  HTTP_HTML,
  HTTP_CGI,
  HTTP_XML,
  HTTP_CSS,
  HTTP_GIF,
  HTTP_PNG,
  HTTP_JPG,
  HTTP_JS,
  HTTP_JAVA,
  HTTP_WAV,
  HTTP_UNKNOWN
} HTTP_FILE_TYPE;
```

Members

Members	Description
HTTP_TXT = 0u	File is a text document
HTTP_HTM	File is HTML (extension .htm)
HTTP_HTML	File is HTML (extension .html)
HTTP_CGI	File is HTML (extension .cgi)
HTTP_XML	File is XML (extension .xml)
HTTP_CSS	File is stylesheet (extension .css)
HTTP_GIF	File is GIF image (extension .gif)
HTTP_PNG	File is PNG image (extension .png)
HTTP_JPG	File is JPG image (extension .jpg)
HTTP_JS	File is java script (extension .js)
HTTP_JAVA	File is java (extension .class)
HTTP_WAV	File is audio (extension .wav)
HTTP_UNKNOWN	File type is unknown

Description

File type definitions

HTTP_IO_RESULT Enumeration

File

http.h

C

```
typedef enum {
  HTTP_IO_DONE = 0u,
```

```
  HTTP_IO_NEED_DATA,
  HTTP_IO_WAITING
} HTTP_IO_RESULT;
```

Members

Members	Description
HTTP_IO_DONE = 0u	Finished with procedure
HTTP_IO_NEED_DATA	More data needed to continue, call again later
HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later

Description

Result states for execution callbacks

HTTP_MODULE_FLAGS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_MODULE_FLAG_DEFAULT = 0x00,
  HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01,
  HTTP_MODULE_FLAG_NO_DELAY = 0x02
} HTTP_MODULE_FLAGS;
```

Members

Members	Description
HTTP_MODULE_FLAG_DEFAULT = 0x00	Default flags value
HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01	Adjust corresponding socket FIFO at run time. Improves throughput when the socket buffers are small.
HTTP_MODULE_FLAG_NO_DELAY = 0x02	Create the HTTP sockets with NO-DELAY option. It will flush data as soon as possible.

Description

HTTP module configuration flags Multiple flags can be OR-ed

HTTP_READ_STATUS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_READ_OK = 0u,
  HTTP_READ_TRUNCATED,
  HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;
```

Members

Members	Description
HTTP_READ_OK = 0u	Read was successful
HTTP_READ_TRUNCATED	Buffer overflow prevented by truncating value
HTTP_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

Description

Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair

HTTP_STATUS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_GET = 0u,
  HTTP_POST,
  HTTP_BAD_REQUEST,
  HTTP_UNAUTHORIZED,
  HTTP_NOT_FOUND,
  HTTP_OVERFLOW,
  HTTP_INTERNAL_SERVER_ERROR,
  HTTP_NOT_IMPLEMENTED,
  HTTP_REDIRECT,
  HTTP_SSL_REQUIRED,
  HTTP_MPFS_FORM,
  HTTP_MPFS_UP,
  HTTP_MPFS_OK,
  HTTP_MPFS_WAIT,
  HTTP_MPFS_ERROR
} HTTP_STATUS;
```

Members

Members	Description
HTTP_GET = 0u	GET command is being processed
HTTP_POST	POST command is being processed
HTTP_BAD_REQUEST	400 Bad Request will be returned
HTTP_UNAUTHORIZED	401 Unauthorized will be returned
HTTP_NOT_FOUND	404 Not Found will be returned
HTTP_OVERFLOW	414 Request-URI Too Long will be returned
HTTP_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
HTTP_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
HTTP_REDIRECT	302 Redirect will be returned
HTTP_SSL_REQUIRED	403 Forbidden is returned, indicating SSL is required
HTTP_MPFS_FORM	Show the MPFS Upload form
HTTP_MPFS_UP	An MPFS Upload is being processed
HTTP_MPFS_OK	An MPFS Upload was successful
HTTP_MPFS_WAIT	An MPFS Upload waiting for the write operation to complete
HTTP_MPFS_ERROR	An MPFS Upload was not a valid image

Description

Supported Commands and Server Response Codes

TCPIP_HTTP_MODULE_CONFIG Structure

File

http.h

C

```
typedef struct {
  uint16_t nConnections;
  uint16_t nTlsConnections;
  uint16_t dataLen;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
  uint16_t tlsSktTxBuffSize;
  uint16_t tlsSktRxBuffSize;
  uint16_t configFlags;
} TCPIP_HTTP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous HTTP connections allowed
uint16_t nTlsConnections;	Not used in the current implementation; Number of simultaneous HTTPS connections allowed
uint16_t dataLen;	size of the data buffer for reading cookie and GET/POST arguments (bytes)

uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default
uint16_t tlsSktTxBuffSize;	Not used in the current implementation; Size of TLS TX buffer for the associated socket; leave 0 for default (min 512 bytes)
uint16_t tlsSktRxBuffSize;	Not used in the current implementation; Size of TLS RX buffer for the associated socket; leave 0 for default (min 512 bytes)
uint16_t configFlags;	a HTTP_MODULE_FLAGS value.

Description

HTTP module dynamic configuration data

Files

Files

Name	Description
http.h	The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server.
http_config.h	HTTP configuration file

Description

This section lists the source and header files used by the library.

http.h

The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server.

Enumerations

	Name	Description
	HTTP_FILE_TYPE	File type definitions
	HTTP_IO_RESULT	Result states for execution callbacks
	HTTP_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	HTTP_READ_STATUS	Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair
	HTTP_STATUS	Supported Commands and Server Response Codes

Functions

	Name	Description
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_CurrentConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionIndexGet	Gets the index of the HTTP connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.

	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket.
	TCPIP_HTTP_GetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_PostExecute	Processes POST form variables and data.
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_Print_varname	Inserts dynamic content into a web page
	TCPIP_HTTP_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.
	TCPIP_HTTP_UserAuthenticate	Performs validation on a specific user name and password.

Macros

	Name	Description
	TCPIP_HTTP_PostReadPair	Reads a name and value pair from a URL encoded string in the TCP buffer.

Structures

	Name	Description
	TCPIP_HTTP_MODULE_CONFIG	HTTP module dynamic configuration data

Types

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection

Description

HTTP Headers for Microchip TCP/IP Stack

The HTTP module runs a web server within the TCP/IP stack. This facilitates an easy method to view status information and control applications using any standard web browser.

File Name

http.h

Company

Microchip Technology Inc.

http_config.h

HTTP configuration file

Macros

	Name	Description
	TCPIP_HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	TCPIP_HTTP_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.
	TCPIP_HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_MAX_HEADER_LEN	Set to length of longest string above
	TCPIP_HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	TCPIP_HTTP_NO_AUTH_WITHOUT_SSL	Uncomment to require secure connection before requesting a password
	TCPIP_HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

	TCPIP_HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTPS_DEFAULT_FILE	Indicate what HTTPS file to serve when no specific one is requested
	TCPIP_STACK_USE_BASE64_DECODE	Authentication requires Base64 decoding Enable basic authentication support

Description

HyperText Transfer Protocol (HTTP) Configuration file

This file contains the HTTP module configuration options

File Name

http_config.h

Company

Microchip Technology Inc.

HTTP Net Module

This section describes the TCP/IP Stack Library HTTP Net Module.

Introduction

TCP/IP Stack Library Hypertext Transfer Protocol (HTTP) Net Module for Microchip Microcontrollers

This library provides the API of the HTTP Net Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The HTTP Net Web Server module allows a development board to act as a Web server. This facilitates an easy method to view status information and control applications using any standard Web browser. It uses the Networking Presentation Layer to integrate with an external encryption services provider (usually wolfSSL) allowing for secure connections.

Using the Library

This topic describes the basic architecture of the HTTP Net TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `http_net.h`

The interface to the HTTP Net TCP/IP Stack library is defined in the `http_net.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the HTTP Net TCP/IP Stack library should include `tcpip.h`.

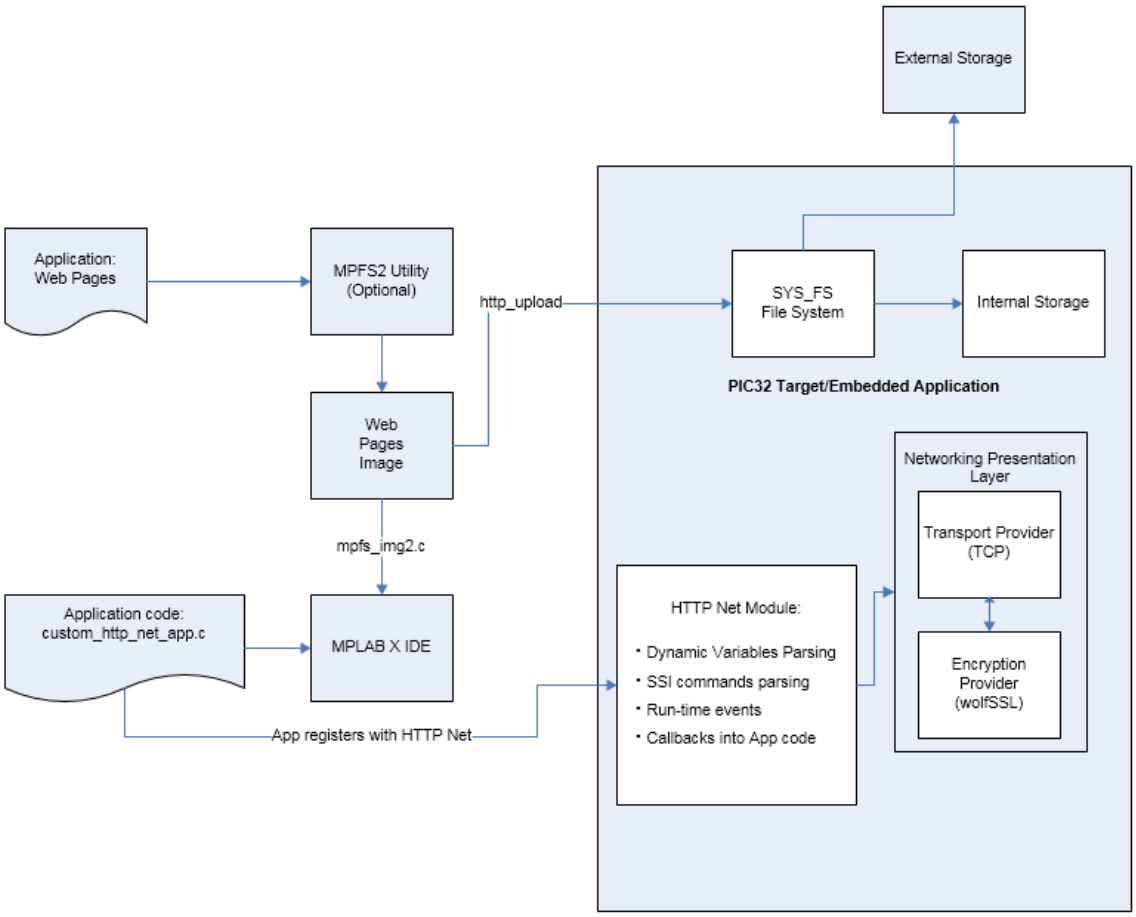
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the HTTP Net TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

HTTP Net 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 HTTP Net module.

Library Interface Section	Description
Functions	This section provides functions for configuring the HTTP Net module.
Data Types and Constants	This section provides various definitions describing this API.

HTTP Net Features

Lists the features of the HTTP Net module.

Description

The HTTP Net module is a HTTP server module that works using the Networking Presentation Layer of the MPLAB Harmony framework. This allows it to support both encrypted and plain text connections.

Some of the key features of the HTTP Net module include:

- The updated module implements HTTP 1.1 persistent connections by default, using the “Transfer-Encoding: chunked” HTTP header. It can be configured to operate with closing the connections like the previous HTTP server.
- The application has to dynamically register its HTTP processing functions with the HTTP Net server using the TCPIP_HTTP_NET_UserHandlerRegister call. The HTTP Net server no longer relies on well known function names.
- The HTTP Net server supports either secure or plain text connections, but not both simultaneously. The specification of a secure or non-secure connection can be easily done by:
  - Selecting the server listening port: 80 or 443
  - Using the module configuration flags
- The HTTP Net module reports run-time events that could be caught by the user application using the “eventReport” registered function that carries a TCPIP_HTTP_NET_EVENT_TYPE enumerated type

Dynamic Variables Processing

Describes dynamic variable processing.

Description

Parsing of the dynamic variables is done at run-time. The web pages could be changed at run-time, without the need for rebuilding the HTTP Net server. This allows the TCP/IP Stack to run as a library and only the application code to be changed accordingly. The application is required to register a "dynamicPrint" with the HTTP Net server. This function will be called at run-time to process the dynamic variables.

The supported syntax is `~var_name(param1, param2, …)~`. When parsing a string like this within a web page, the HTTP Net server will invoke the "dynamicPrint" function using a TCPIP_HTTP_DYN_VAR_DCPT data structure that specifies the dynamic variable name, its number and type of parameters as well as each parameter value.

For example, the variable `~myVariable(2,6)~` will generate the "dynamicPrint" call with the following parameters:

- `varDcpt.dynName = "myVariable";`
- `varDcpt.nArgs = 2;`
- `varDcpt.dynArgs->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32;`
- `varDcpt.dynArgs->argInt32 = 2;`
- `(varDcpt.dynArgs + 1)->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32;`
- `(varDcpt.dynArgs + 1)->argInt32 = 6;`

String and int32_t variable types are currently supported.

The application needs to return a TCPIP_HTTP_DYN_PRINT_RES result specifying if it's done with processing or it needs to be called again.

Applications no longer have direct access to the underlying transport socket for a HTTP connection. All of the data write operations need to go through the required API function: TCPIP_HTTP_NET_DynamicWrite. This function actually just attaches the data to be written to the connection, using data descriptors. The actual output occurs only when all data is gathered and the data size is known.

Buffers used in a TCPIP_HTTP_NET_DynamicWrite need to be persistent until the moment when the output is generated. RAM buffers are supported by using a "dynamicAck" (TCPIP_HTTP_NET_DynAcknowledge) type function. Once the output operation is completed the HTTP Net module will call back into the application indicating that the corresponding buffer is no longer used and can be freed/reused.

In situations where the dynamic variable print function needs to perform additional write operations, or simply needs to be called again, it must a special result code: TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN. Typically this is used when outputting large amounts of data that cannot fit into one single buffer write operation or when the data is not available all at once.

Include Files Processing

Provides information on include file processing.

Description

The HTTP Net module dynamically parses the web pages and the file inclusion is done by the module itself, completely transparent for the application.

File Processing

Provides information on file processing.

Description

File processing consists of the following:

- An included file can include other files that have dynamic variables
- Application can include a file as part of its dynamic variable processing using the TCPIP_HTTP_NET_ConnectionFileInclude()API. That file can include other files.
- The only limit to the depth of inclusion is the memory resources and the TCPIP_HTTP_NET_MODULE_CONFIG:: maxRecurseLevel setting. A TCPIP_HTTP_NET_EVENT_DEPTH_ERROR event will be generated if the maximum depth is exceeded at run-time.

SSI Processing

Provides information on SSI processing.

Description

At run-time, the HTTP Net server processes a subset of the Server Side Includes (SSI) commands. SSI is a portable way to add small amounts of dynamic content on web pages in a standard way that is supported by most HTTP servers. Currently the `#include`, `#set` and `#echo` commands are supported. Other commands will be eventually added. This allows a better compatibility with other existing HTTP servers and is targeted at the removal of proprietary extensions.

Whenever executing an SSI command within a web page, the HTTP Net server will notify the application by using the registered SSI callback:

ssiNotify. This callback contains a TCPIP_HTTP_SSI_NOTIFY_DCPT pointer to a structure describing the SSI context needed for processing:

- File name the SSI command belongs to
- The SSI command line
- Number of SSI attributes and their values, etc.

The application can do any modifications it chooses to and returns a value instructing the HTTP Net server if the processing of the command is necessary or not.

SSI include Command

The SSI Include command allows the dynamic inclusion of a file. The supported syntax is:

`<!--#include virtual="file_name" -->` or `<!--#include file="file_name" -->`.

Currently the arguments for both “virtual” and “file” commands are passed unaltered to the SYS_FS, so they behave identically. However it is recommended that the original SSI significance for these keywords should be maintained:

- Use “virtual” for specifying a URL relative to the document being server
- Use “file” for a file path, relative to the current directory (it cannot be an absolute path)

The ~inc:file_name~ keyword is maintained for backward compatibility. However the SSI include command should be preferred.

SSI set Command

The SSI set command allows to dynamically set a SSI variable value. The supported syntax is:

`<!--#set var="v_name" value="v_value" -->`.

String or integer variables are supported. Variable reference is also supported: `<!--#set var="n_name" value="$otherVar" -->`. This command will create or update the variable n_name to have the value of the variable otherVar, if it exists.

A new SSI variable will be created if a variable having the name v_name does not exist. If the variable v_name already exists, it will have its value updated as the result of this command.

An existing variable can be deleted using the empty value set command:

`<!--#set var="v_name" value="" -->`.

SSI echo Command

The SSI echo command allows the dynamic print of a SSI variable. The supported syntax is:

`<!--#echo var="v_name" -->`.

If the application ssiNotify exists, the HTTP Net server will call it and the application may choose to change the current value dynamically. If ssiNotify returns false, the HTTP Net server will display the current value of the variable v_name as part of the current page.

The SSI API can be used to evaluate or change the current value of the SSI variables:

- TCPIP_HTTP_NET_SSIVariableGet
- TCPIP_HTTP_NET_SSIVariableSet
- TCPIP_HTTP_NET_SSIVariableDelete

The maximum number of SSI variables is under the control of the application by using the configuration parameter: TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER.

mpfs2.jar Utility

Provides information on the option mpfs2 utility.

Description

Dynamic Variable Processing

The HTTP Net server does not use the DynRecord.bin and FileRcrd.bin files for processing the dynamic variables. The parsing and processing of dynamic variables is done at run-time. Therefore, the mpfs2.jar utility is no longer necessary, other than to generate a mpfs_img2.c image file for the NVM MPFS image.

Application Custom Support Files

The application custom support files generated with MHC have new names: custom_http_net_app.c and http_net_print.c. However, the http_net_print.c file is not generated by the mpfs2 utility and is maintained only for easy comparison between HTTP and HTTP Net application processing. The custom_http_net_app.c file is entirely generated using a MHC template file and it is not dynamically updated in any way by the mpfs2 utility.

Generated File Name

Currently the name of the generated file for MPFS image is maintained unchanged: mpfs_img2.c.

MPFS Image Generation for Internal NVM Storage

The mpfs2 utility can still be used to generate the MPFS image for internal NVM storage. It can also be useful because it parses the web pages and comes out with the `http_print.c` file that contains the list of TCPIP_HTTP_Print functions. This can be helpful in gathering info about the dynamic variables that are contained within the web pages.

Note: For demonstration applications that use SSI, the file inclusion is now done in a standard way using HTML (i.e., `.htm`) files. Therefore, when generating the image, `*.htm` must be added to *Advanced Settings* > *Do Not Compress*.

Configuring the Library

Macros

	Name	Description
	TCPIP_HTTP_NET_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_NET_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE	Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.
	TCPIP_HTTP_NET_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER	maximum number of arguments for a dynamic variable
	TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER	how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
	TCPIP_HTTP_NET_DYNVAR_MAX_LEN	maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_NET_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.
	TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE	Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.
	TCPIP_HTTP_NET_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_NET_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_NET_MAX_HEADER_LEN	The length of longest header string that can be parsed
	TCPIP_HTTP_NET_MAX_RECURSE_LEVEL	The maximum depth of recursive calls for serving a web page: • no dynvars files: 1 • file including a file: 2 • if the include file includes another file: +1 • if a dyn variable: +1 Default value is 3;
	TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE	Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes
	TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

	TCPIP_HTTP_NET_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NET_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_NET_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_NET_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_NET_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTP_NET_CHUNK_RETRIES	retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted
	TCPIP_HTTP_NET_CHUNKS_NUMBER	number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections
	TCPIP_HTTP_NET_DYNVAR_PROCESS	This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0
	TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES	retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES	Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE	Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER	Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of... more
	TCPIP_HTTP_NET_FILENAME_MAX_LEN	maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator
	TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER	maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense
	TCPIP_HTTP_NET_SSI_CMD_MAX_LEN	maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE	message to echo when echoing a not found variable
	TCPIP_HTTP_NET_SSI_PROCESS	This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0
	TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER	number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically

	TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH	maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable
	TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH	maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.
	TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER	maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.

Description

The configuration of the HTTP Net TCP/IP Stack is based on the file `tcpip_config.h` (which may include `http_net_config.h`).

This header file contains the configuration selection for the HTTP Net TCP/IP Stack. Based on the selections made, the HTTP Net TCP/IP Stack 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.

TCPIP_HTTP_NET_CACHE_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CACHE_LEN ("600")
```

Description

Max lifetime (sec) of static responses as string

TCPIP_HTTP_NET_CONFIG_FLAGS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CONFIG_FLAGS 0
```

Description

Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values

TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE 200
```

Description

Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.

TCPIP_HTTP_NET_DEFAULT_FILE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTP file to serve when no specific one is requested

TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER 10
```

Description

maximum number of arguments for a dynamic variable

TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER 10
```

Description

how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool

TCPIP_HTTP_NET_DYNVAR_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_MAX_LEN 50
```

Description

maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read

TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE
```

Description

Configure MPFS over HTTP updating Comment this line to disable updating via HTTP

TCPIP_HTTP_NET_FILE_UPLOAD_NAME Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_UPLOAD_NAME "mpfsupload"
```

Description

This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.

TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE 512
```

Description

Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.

Remarks

if upload is enabled (TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE) then it is recommended for the peek buffer size to match the size of the socket RX buffer: TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE Note that this is an automatic buffer (created on the stack) and enough stack space should be provided for the application.

TCPIP_HTTP_NET_MAX_CONNECTIONS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_CONNECTIONS (4)
```

Description

Maximum numbers of simultaneous supported HTTP connections.

TCPIP_HTTP_NET_MAX_DATA_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_DATA_LEN (100u)
```

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

TCPIP_HTTP_NET_MAX_HEADER_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_HEADER_LEN (15u)
```

Description

The length of longest header string that can be parsed

TCPIP_HTTP_NET_MAX_RECURSE_LEVEL Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_RECURSE_LEVEL 3
```

Description

The maximum depth of recursive calls for serving a web page:

- no dynvars files: 1
- file including a file: 2
- if the include file includes another file: +1
- if a dyn variable: +1

Default value is 3;

TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE 300
```

Description

Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes

TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE 2048
```

Description

Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE 2048
```

Description

Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_NET_TASK_RATE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_TASK_RATE 33
```

Description

The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_HTTP_NET_TIMEOUT Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_TIMEOUT (45u)
```

Description

Max time (sec) to await more data before timing out and disconnecting the socket

TCPIP_HTTP_NET_USE_AUTHENTICATION Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_AUTHENTICATION
```

Description

Enable basic authentication support

TCPIP_HTTP_NET_USE_COOKIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_COOKIES
```

Description

Enable cookie support

TCPIP_HTTP_NET_USE_POST Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_POST
```

Description

Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

TCPIP_HTTP_NET_CHUNK_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CHUNK_RETRIES 10
```

Description

retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted

TCPIP_HTTP_NET_CHUNKS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CHUNKS_NUMBER 10
```

Description

number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections

TCPIP_HTTP_NET_DYNVAR_PROCESS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_PROCESS 1
```

Description

This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0

TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES 10
```

Description

retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again

TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES 10
```

Description

Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.

TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE 512
```

Description

Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!

TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER 4
```

Description

Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of maximum files that can be open simultaneously: i.e. for file1 ->include file2 -> include file3 you'll need >= 3 file process buffers

TCPIP_HTTP_NET_FILENAME_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILENAME_MAX_LEN 25
```

Description

maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator

TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER 4
```

Description

maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense

TCPIP_HTTP_NET_SSI_CMD_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_CMD_MAX_LEN 100
```

Description

maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read

TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE "SSI Echo - Not Found: "
```

Description

message to echo when echoing a not found variable

TCPIP_HTTP_NET_SSI_PROCESS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_PROCESS 1
```

Description

This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0

TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER 2
```

Description

number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically

TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH 10
```

Description

maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable

TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH 10
```

Description

maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.

TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER 13
```

Description

maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.

Building the Library

This section lists the files that are available in the HTTP Net TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/http_net.c`	HTTP Server 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 HTTP Net module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) Functions

	Name	Description
	TCPIP_HTTP_NET_ActiveConnectionCountGet	Gets the number of active (and inactive) connections.
	TCPIP_HTTP_NET_ConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_NET_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_NET_ConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionByteCountGet	Returns how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_NET_ConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_NET_ConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDiscard	Discards any pending data in the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionFileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_NET_ConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_NET_ConnectionFileInclude	Writes a file to the HTTP connection.
	TCPIP_HTTP_NET_ConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionFlush	Immediately transmits all connection pending TX data.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionGetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_NET_ConnectionPostNameRead	Reads a name from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionPostValueRead	Reads a value from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionNetHandle	Returns the network handle of the current connection.
	TCPIP_HTTP_NET_ConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_NET_ConnectionPeek	Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.
	TCPIP_HTTP_NET_ConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_NET_ConnectionPostExecute	Processes POST form variables and data.
	TCPIP_HTTP_NET_ConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_NET_ConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_NET_ConnectionRead	Reads an array of data bytes from a connection's RX buffer.
	TCPIP_HTTP_NET_ConnectionReadBufferSize	Returns the size of the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionReadIsReady	Determines how many bytes can be read from the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_NET_ConnectionStringFind	Helper to find a string of characters in an incoming connection buffer.
	TCPIP_HTTP_NET_ConnectionUserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_NET_ConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_NET_DynAcknowledge	Dynamic variable acknowledge function
	TCPIP_HTTP_NET_DynamicWrite	Writes a data buffer to the current connection
	TCPIP_HTTP_NET_DynamicWriteString	Helper for writing a string within a dynamic variable context.
	TCPIP_HTTP_NET_DynPrint	Inserts dynamic content into a web page
	TCPIP_HTTP_NET_EventReport	Reports an event that occurred in the processing of a HTTP web page
	TCPIP_HTTP_NET_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_NET_URLDecode	Parses a string from URL encoding to plain text.
	TCPIP_HTTP_NET_UserHandlerDeregister	Deregisters a previously registered HTTP user handler.
	TCPIP_HTTP_NET_UserHandlerRegister	Registers a user callback structure with the HTTP server
	TCPIP_HTTP_NET_ConnectionDataBufferSizeGet	Returns the size of the connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDynamicDescriptors	Returns the number of dynamic variable descriptors
	TCPIP_HTTP_NET_SSINotification	Reports an SSI processing event that occurs in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSIVariableDelete	Function to delete an SSI variable.

	TCPIP_HTTP_NET_SSIVariableGet	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableGetByIndex	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableSet	Function to set an SSI variable.
	TCPIP_HTTP_NET_SSIVariablesNumberGet	Function to get the number of the current SSI variables.
	TCPIP_HTTP_NET_ConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_NET_ConnectionIndexGet	Gets the index of the HTTP connection.

b) Data Types and Constants

	Name	Description
	_tag_TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_DYN_ARG_DCPT	HTTP dynamic argument descriptor.
	TCPIP_HTTP_DYN_ARG_TYPE	HTTP supported dynamic variables argument types.
	TCPIP_HTTP_DYN_PRINT_RES	Dynamic print result when a dynamic variable print callback function returns;
	TCPIP_HTTP_DYN_VAR_DCPT	HTTP dynamic variable descriptor.
	TCPIP_HTTP_DYN_VAR_FLAGS	HTTP supported dynamic variables flags.
	TCPIP_HTTP_NET_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	TCPIP_HTTP_NET_EVENT_TYPE	HTTP reported run-time events.
	TCPIP_HTTP_NET_IO_RESULT	Result states for execution callbacks
	TCPIP_HTTP_NET_MODULE_CONFIG	HTTP module dynamic configuration data
	TCPIP_HTTP_NET_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	TCPIP_HTTP_NET_READ_STATUS	Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair
	TCPIP_HTTP_NET_STATUS	Supported Commands and Server Response Codes
	TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_NET_USER_HANDLE	HTTP user handle.
	TCPIP_HTTP_NET_ConnectionPostReadPair	Reads a name and value pair from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_SSI_ATTR_DCPT	HTTP SSI attribute descriptor.
	TCPIP_HTTP_SSI_NOTIFY_DCPT	HTTP SSI notification descriptor.

Description

This section describes the Application Programming Interface (API) functions of the HTTP Net module.

Refer to each section for a detailed description.

a) Functions

TCPIP_HTTP_NET_ActiveConnectionCountGet Function

Gets the number of active (and inactive) connections.

File

http_net.h

C

```
int TCPIP_HTTP_NET_ActiveConnectionCountGet(int* pOpenCount);
```

Returns

The number of active/total connections.

Description

This function will return the number of active and total opened HTTP connections at the current time.

Remarks

The value returned by this function is informational only. The number of active connections changes dynamically.

Preconditions

None.

Example

```
int nConns;

nConns = TCPIP_HTTP_NET_ActiveConnectionCountGet(0);
```

Parameters

Parameters	Description
pOpenCount	address to store the number of total HTTP opened connections Could be NULL if not needed

Function

int TCPIP_HTTP_NET_ActiveConnectionCountGet(int* pOpenCount);

TCPIP_HTTP_NET_ConnectionByteCountDec Function

Decrements the connection byte count.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionByteCountDec(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function decrements the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count reduction

Function

void TCPIP_HTTP_NET_ConnectionByteCountDec(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_NET_ArgGet Function

Locates a form field value in a given data array.

File

http_net.h

C

```
const uint8_t* TCPIP_HTTP_NET_ArgGet(const uint8_t* cData, const uint8_t* cArg);
```

Returns

A pointer to the argument value, or NULL if not found.

Description

This function searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of

null-terminated strings, followed by an additional null terminator.

Remarks

None.

Preconditions

The data array has a valid series of null terminated name/value pairs.

Example

```
TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const TCPIP_HTTP_DYN_VAR_DCPT*
                        varDcpt, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
{
    const uint8_t *ptr;

    ptr = TCPIP_HTTP_NET_ArgGet(TCPIP_HTTP_NET_ConnectionDataBufferGet(connHandle),
                                (const uint8_t*)"name");
    if(ptr == 0)
    {
        ptr = "not set";
    }
    else
    {
        strncpy(myBuffer, ptr, sizeof(myBuffer));
        ptr = myBuffer;
    }

    TCPIP_HTTP_NET_DynamicWrite(varDcpt, ptr, strlen(ptr), false);

}
```

Parameters

Parameters	Description
cData	the buffer to search
cArg	the name of the argument to find

Function

const uint8_t* TCPIP_HTTP_NET_ArgGet(const uint8_t* cData, const uint8_t* cArg)

TCPIP_HTTP_NET_ConnectionByteCountSet Function

Sets how many bytes have been read so far.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionByteCountSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function sets the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

byteCount	byte count to be set

Function

void TCPIP_HTTP_NET_ConnectionByteCountSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_NET_ConnectionByteCountGet Function

Returns how many bytes have been read so far.

File

http_net.h

C

```
uint32_t TCPIP_HTTP_NET_ConnectionByteCountGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Current connection byte count, how many bytes have been read so far.

Description

This function returns the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Example

```
switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
    case SM_CFG_SNMP_READ_NAME:
        // If all parameters have been read, end
        if(TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle) == 0u)
        {
            return TCPIP_HTTP_NET_IO_RES_DONE;
        }
     .
     .
     .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_NET_ConnectionByteCountGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionCallbackPosSet Function

Sets the callback position indicator.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionCallbackPosSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t callbackPos);
```

Returns

None.

Description

This function will set the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables. When set to a value != 0, it indicates to the HTTP server that the application has more pending processing that needs to be done.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
connHandle	HTTP connection handle
callbackPos	new connection callback position value

Function

void TCPIP_HTTP_NET_ConnectionCallbackPosSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t callbackPos)

TCPIP_HTTP_NET_ConnectionCallbackPosGet Function

Returns the callback position indicator.

File

http_net.h

C

```
uint32_t TCPIP_HTTP_NET_ConnectionCallbackPosGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Callback position indicator for connection defined by connHandle.

Description

This function will return the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_NET_ConnectionCallbackPosGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionDataBufferGet Function

Returns pointer to connection general purpose data buffer.

File

http_net.h

C

```
uint8_t* TCPIP_HTTP_NET_ConnectionDataBufferGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Pointer to the connection's general purpose data buffer.

Description

This function returns a pointer to the HTTP connection internal data buffer. This gives access to the application to the data that's stored in the HTTP connection buffer.

Remarks

None.

Preconditions

None.

Example

```
TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
   const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,
   const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
{
    const uint8_t *ptr;

    ptr = TCPIP_HTTP_NET_ArgGet(TCPIP_HTTP_NET_ConnectionDataBufferGet
                              (connHandle), (const uint8_t*)"name");
    if(ptr == 0)
    {
        ptr = "not set";
    }
    else
    {
        strncpy(myBuffer, ptr, sizeof(myBuffer));
        ptr = myBuffer;
    }
    TCPIP_HTTP_NET_DynamicWrite(varDcpt, ptr, strlen(ptr), false);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t* TCPIP_HTTP_NET_ConnectionDataBufferGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionDiscard Function

Discards any pending data in the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionDiscard(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The number of bytes that have been discarded from the RX buffer.

Description

This function discards data from the connection RX buffer.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionDiscard(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionFileAuthenticate Function

Determines if a given file name requires authentication

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionFileAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile,
const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- <= 0x79 - valid authentication is required
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved for the current connection and can be read using TCPIP_HTTP_NET_ConnectionIsAuthorizedGet. It will be available to future callbacks, including TCPIP_HTTP_NET_ConnectionUserAuthenticate and any of the TCPIP_HTTP_NET_ConnectionGetExecute, TCPIP_HTTP_NET_ConnectionPostExecute, or TCPIP_HTTP_NET_Print_varname callbacks.

Remarks

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file being requested
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

uint8_t TCPIP_HTTP_NET_ConnectionFileAuthenticate

(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

const char* cFile, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionFileGet Function

Get handle to current connection's file.

File

http_net.h

C

```
SYS_FS_HANDLE TCPIP_HTTP_NET_ConnectionFileGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Handle to File System file belonging to the connection defined by connHandle

Description

This function returns the handle of the current HTTP connection file.

Remarks

None.

Preconditions

None.

Example

```
uint8_t myBuff[20];

// Get the file handle and read from that file
SYS_FS_FileRead(myBuff, sizeof(myBuff), TCPIP_HTTP_NET_ConnectionFileGet(connHandle));
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

FILE_HANDLE TCPIP_HTTP_NET_ConnectionFileGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionFileInclude Function

Writes a file to the HTTP connection.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_ConnectionFileInclude(HTTP_CONN_HANDLE connHandle, const char* fileName);
```

Returns

true if the call succeeded, false if the file could not be opened

Description

This function allows an entire file to be included as part of a dynamic variable processing. This reduces unneeded duplication of visual elements such as headers, menus, etc.

It is not meant for the processing of dynamic variables that include files by using the "inc" keyword (see the Remarks section below).

Remarks

Should be called from within a dynamic variable processing function context. The function allows to insert a file, as part of the dynamic variable processing.

The included file can contain dynamic variables. However there's a limit for recursive calls (see TCPIP_HTTP_NET_MAX_RECURSE_LEVEL).

The file is just added to the processing queue. Returning true does not mean that the whole file has been already sent to the connection.

If the function returns true but an error occurs during the file processing an event will be reported using the TCPIP_HTTP_NET_EventReport callback.

If the function returns false an event will be reported using the TCPIP_HTTP_NET_EventReport callback with additional info.

Please note that the processing of HTTP dynamic keywords in the HTML code such as `~inc:filename.ext~` is processed internally by the HTTP module! For such a dynamic variable, control is not passed to the user.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

fileName	the name of the file to be included

Function

bool TCPIP_HTTP_NET_ConnectionFileInclude(HTTP_CONN_HANDLE connHandle, const char* fileName)

TCPIP_HTTP_NET_ConnectionHasArgsGet Function

Checks whether there are get or cookie arguments.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionHasArgsGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The current value of the connection hasArgs.

Description

The function will get the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
uint8_t hasArgs = TCPIP_HTTP_NET_ConnectionHasArgsGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_NET_ConnectionHasArgsGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionFlush Function

Immediately transmits all connection pending TX data.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionFlush(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

- number of flushed bytes
- 0 if no flushed bytes or some error

Description

This function calls the transport layer's flush function

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionFlush(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionIsAuthorizedGet Function

Gets the authorized state for the current connection.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

A uint8_t representing the authorization status.

Description

This function returns the authorization status for the current HTTP connection. This is one of the values returned by the TCPIP_HTTP_NET_ConnectionFileAuthenticate function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t isAuth;

isAuth = TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionGetExecute Function

Processes GET form field variables and cookies.

File

http_net.h

C

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionGetExecute(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- TCPIP_HTTP_NET_IO_RES_DONE - application is done processing
- TCPIP_HTTP_NET_IO_RES_NEED_DATA - this value may not be returned because more data will not become available
- TCPIP_HTTP_NET_IO_RES_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, the connection data buffer (see TCPIP_HTTP_NET_ConnectionDataBufferGet) contains sequential name/value pairs of strings representing the data received. In this format, TCPIP_HTTP_NET_ArgGet can be used to search for specific variables in the input. If

data buffer space associated with this connection is required, connection data buffer may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_NET_ConnectionPostExecute and any TCPIP_HTTP_NET_Print_varname dynamic substitutions.

This function may also issue redirections by setting the connection data buffer to the destination file name or URL, and the connection httpStatus (TCPIP_HTTP_NET_ConnectionStatusSet()) to TCPIP_HTTP_NET_STAT_REDIRECT.

Finally, this function may set cookies. Set connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive) and then set the connection hasArgs (TCPIP_HTTP_NET_ConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments.

This function may NOT write to the network transport buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_NET_ConnectionCallbackPosGet) or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionGetExecute
(                TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const                TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
```

TCPIP_HTTP_NET_ConnectionPostNameRead Function

Reads a name from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostNameRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint8_t* cData, uint16_t wLen);
```

Returns

- TCPIP_HTTP_NET_READ_OK - name was successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire name could not fit in the buffer, so the value was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire name was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a name from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

Remarks

None.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostNameRead (TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_NET_ConnectionHasArgsSet Function

Sets whether there are get or cookie arguments.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionHasArgsSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t args);
```

Returns

None.

Description

The function sets the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
else if(!memcmp(filename, "cookies.htm", 11))
{
    TCPIP_HTTP_NET_ConnectionHasArgsSet(connHandle, true);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
args	boolean if there are arguments or not

Function

void TCPIP_HTTP_NET_ConnectionHasArgsSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t args)

TCPIP_HTTP_NET_ConnectionPostSmGet Function

Get the POST state machine state.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionPostSmGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

16-bit integer POST state machine state.

Description

This function returns the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:
     .
     .
     .
    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
     .
     .
     .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionPostSmGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionIsAuthorizedSet Function

Sets the authorized state for the current connection.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t auth);
```

Returns

None.

Description

This function sets the authorization status for the current HTTP connection. This has to be one of the values in the set returned by the TCPIP_HTTP_NET_ConnectionFileAuthenticate function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t auth = 0x80;

TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(connHandle, auth);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
auth	new authorization state

Function

void TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t auth)

TCPIP_HTTP_NET_ConnectionPostValueRead Function

Reads a value from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostValueRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint8_t* cData, uint16_t wLen);
```

Returns

- TCPIP_HTTP_NET_READ_OK - value was successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire value could not fit in the buffer, so the value was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire value was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a value from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If pHttpCon->byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

Remarks

None.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the value once it is read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostValueRead (TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_NET_ConnectionNetHandle Function

Returns the network handle of the current connection.

File

http_net.h

C

```
TCPIP_NET_HANDLE TCPIP_HTTP_NET_ConnectionNetHandle(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The connection network handle.

Description

This function returns the network handle over which the current HTTP connection communicates.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

TCPIP_NET_HANDLE TCPIP_HTTP_NET_ConnectionNetHandle(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionSocketGet Function

Get the socket for the current connection.

File

http_net.h

C

```
NET_PRES_SKT_HANDLE_T TCPIP_HTTP_NET_ConnectionSocketGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

NET_PRES_SKT_HANDLE_T for the connection defined by connHandle.

Description

The function returns the network transport socket of the specified HTTP connection. The user gets access to the connection socket which it can use for debugging or directly sending/reading data.

Remarks

This function gives direct access to the underlying transport socket. It is meant for test/advanced usage only. The regular connection functions should be used for manipulation of the connection data. Using the socket directly for data manipulation will disrupt the HTTP server functionality.

Preconditions

None.

Example

```
uint32_t byteCount;
int sktRxSize;

byteCount = TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle);
sktRxSize = TCPIP_HTTP_NET_ConnectionReadBufferSize(connHandle);

if(byteCount > sktRxSize)
{   // Configuration Failure
    TCPIP_HTTP_NET_ConnectionStatusSet(connHandle, TCPIP_HTTP_NET_STAT_REDIRECT);
    return TCPIP_HTTP_NET_IO_RES_DONE;
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

NET_PRES_SKT_HANDLE_T TCPIP_HTTP_NET_ConnectionSocketGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionPeek Function

Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionPeek(TCPIP_HTTP_NET_CONN_HANDLE connHandle, void * buffer, uint16_t size);
```

Description

The function allows peeking into the connection buffer. The data will still be available for a next read operation.

Remarks

None

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle
buffer	Destination to write the peeked data bytes
size	Length of bytes to peek from the connection RX buffer

Function

uint16_t TCPIP_HTTP_NET_ConnectionPeek(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
void * buffer, uint16_t size);

TCPIP_HTTP_NET_ConnectionStatusGet Function

Gets HTTP status.

File

http_net.h

C

```
TCPIP_HTTP_NET_STATUS TCPIP_HTTP_NET_ConnectionStatusGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

A TCPIP_HTTP_NET_STATUS value.

Description

This function returns the current HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
TCPIP_HTTP_NET_STATUS currStat =  TCPIP_HTTP_NET_ConnectionStatusGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

TCPIP_HTTP_NET_STATUS TCPIP_HTTP_NET_ConnectionStatusGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionPostExecute Function

Processes POST form variables and data.

File

http_net.h

C

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionPostExecute(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- TCPIP_HTTP_NET_IO_RES_DONE - application is done processing
- TCPIP_HTTP_NET_IO_RES_NEED_DATA - more data is needed to continue, and this function should be called again later
- TCPIP_HTTP_NET_IO_RES_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the network transport buffer. The connection byteCount (see TCPIP_HTTP_NET_ConnectionByteCountGet will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the network transport buffer, the application must call TCPIP_HTTP_NET_ConnectionRead in order to retrieve bytes. When this is done, connection byteCount MUST be updated to reflect how many bytes now remain.

In general, data submitted from web forms via POST is URL encoded. The TCPIP_HTTP_NET_URLDecode function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, the connection data buffer (see TCPIP_HTTP_NET_ConnectionDataBufferGet) may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_NET_ConnectionPostExecute and any TCPIP_HTTP_NET_Print_varname dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting the connection data buffer to the destination file or URL, and the connection httpStatus (TCPIP_HTTP_NET_ConnectionStatusSet) to TCPIP_HTTP_NET_STAT_REDIRECT.

Finally, this function may set cookies. Set the connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive), and then set the connection hasArgs (TCPIP_HTTP_NET_ConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called when the request method is POST, and is only used when TCPIP_HTTP_NET_USE_POST is defined.

This method may NOT write to the network transport buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_NET_ConnectionCallbackPosGet) or connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionPostExecute
(        TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const        TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
```

TCPIP_HTTP_NET_ConnectionUserDataGet Function

Gets the user data parameter for the current connection.

File

http_net.h

C

```
const void* TCPIP_HTTP_NET_ConnectionUserDataGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

User data that's stored as part of the connection.

Description

This function returns the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It can be set by the user with TCPIP_HTTP_NET_ConnectionUserDataSet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

myConnData = (uint32_t)TCPIP_HTTP_NET_ConnectionUserDataGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

const void* TCPIP_HTTP_NET_ConnectionUserDataGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionPostSmSet Function

Set the POST state machine state.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionPostSmSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint16_t state);
```

Returns

None.

Description

This function sets the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
uint8_t* httpDataBuff;
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
```

```
        // Find the name
        case SM_POST_LCD_READ_NAME:

            // Read a name
            if(TCPIP_HTTP_NET_ConnectionPostNameRead(connHandle, httpDataBuff,
               TCPIP_HTTP_NET_MAX_DATA_LEN) == TCPIP_HTTP_NET_READ_INCOMPLETE)
                return TCPIP_HTTP_NET_IO_RES_NEED_DATA;

            TCPIP_HTTP_NET_ConnectionPostSmSet(connHandle, SM_POST_LCD_READ_VALUE);
            // No break...continue reading value

        // Found the value, so store the LCD and return
        case SM_POST_LCD_READ_VALUE:
         .
         .
         .
    }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
state	16 bit integer state for POST state machine

Function

void TCPIP_HTTP_NET_ConnectionPostSmSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint16_t state)

TCPIP_HTTP_NET_ConnectionRead Function

Reads an array of data bytes from a connection's RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle, void * buffer, uint16_t size);
```

Returns

The number of bytes read from the socket. If less than len, the connection RX buffer became empty or the underlying socket is not connected.

Description

This function reads an array of data bytes from the connection RX buffer. The data is removed from the FIFO in the process.

Remarks

If the supplied buffer is null, the data is simply discarded.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle
buffer	The pointer to the array to store data that was read
size	The number of bytes to be read.

Function

uint16_t TCPIP_HTTP_NET_ConnectionRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle, void * buffer, uint16_t size);

TCPIP_HTTP_NET_ConnectionReadBufferSize Function

Returns the size of the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionReadBufferSize(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The size of the connection RX buffer.

Description

This function discards data from the connection RX buffer.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionReadBufferSize(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionReadIsReady Function

Determines how many bytes can be read from the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionReadIsReady(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The number of bytes available to be read from the connection RX buffer.

Description

This function determines how many bytes can be read from the connection RX buffer.

Remarks

When using an encrypted connection the number of available unencrypted bytes may turn out to be different than what this function returns.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionReadIsReady(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionStatusSet Function

Sets HTTP status.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionStatusSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_STATUS stat);
```

Returns

None.

Description

Allows write access to the HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
byteCount = TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle);
int sktRxSize;

sktRxSize = TCPIP_HTTP_NET_ConnectionReadBufferSize(connHandle);

if(byteCount > sktRxSize)
{   // Configuration Failure
    // 302 Redirect will be returned
    TCPIP_HTTP_NET_ConnectionStatusSet(connHandle, TCPIP_HTTP_NET_STAT_REDIRECT);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
stat	new TCPIP_HTTP_NET_STATUS enumeration value.

Function

void TCPIP_HTTP_NET_ConnectionStatusSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
TCPIP_HTTP_NET_STATUS stat)

TCPIP_HTTP_NET_ConnectionStringFind Function

Helper to find a string of characters in an incoming connection buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionStringFind(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* str,
uint16_t startOffs, uint16_t searchLen);
```

Returns

- If string was found - a zero-indexed position in the RX buffer of the string occurrence
- 0xFFFF - Search array not found

Description

This function searches for an ASCIIZ string in the RX buffer of the connection. It does the search by performing a peek operation in the RX buffer, (i.e., the RX data in the buffer is not consumed and it is available for further read operations). It works for both encrypted and unencrypted connections.

Remarks

Note that the search will fail if there's more data in the TCP socket than could be read at once.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	HTTP connection handle
str	0 terminated ASCII string to search for
startOffs	offset in the RX buffer to start the search from
searchLen	if !0 it is the length of buffer to search into (starting from startOffs); if 0, the whole buffer is searched

Function

uint16_t TCPIP_HTTP_NET_ConnectionStringFind(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

const char* str, uint16_t startOffs, uint16_t searchLen);

TCPIP_HTTP_NET_ConnectionUserAuthenticate Function

Performs validation on a specific user name and password.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionUserAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser,
const char* cPass, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- <= 0x79 - the credentials were rejected
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if the user name and password supplied by the client are acceptable for this resource. This callback function can thus to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The value returned by this function is stored in the corresponding connection data and will be available with TCPIP_HTTP_NET_ConnectionIsAuthorizedGet in any of the TCPIP_HTTP_NET_ConnectionGetExecute, TCPIP_HTTP_NET_ConnectionPostExecute, or TCPIP_HTTP_NET_Print_varname callbacks.

Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cUser	the user name supplied by the client
cPass	the password supplied by the client
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

uint8_t TCPIP_HTTP_NET_ConnectionUserAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE

connHandle, const char* cUser, const char* cPass,

const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionUserDataSet Function

Sets the user data parameter for the current connection.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionUserDataSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* uData);
```

Returns

None.

Description

This function will set the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It is available to the user by calling TCPIP_HTTP_NET_ConnectionUserDataGet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

TCPIP_HTTP_NET_ConnectionUserDataSet(connHandle, (const void*)myConnData);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
uData	user supplied data

Function

void TCPIP_HTTP_NET_ConnectionUserDataSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* uData)

TCPIP_HTTP_NET_DynAcknowledge Function

Dynamic variable acknowledge function

File

http_net.h

C

```
void TCPIP_HTTP_NET_DynAcknowledge(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. This function reports that a buffer that was passed as part of a dynamic variable callback (dynamicPrint) was processed and can be reused, freed, etc.

Remarks

The function is called only when TCPIP_HTTP_NET_DynamicWrite or TCPIP_HTTP_NET_DynamicWriteString was specified with a true needAck parameter.

The function is called from withing HTTP context. It should be kept as short as possible.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
buffer	the processed buffer
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

void TCPIP_HTTP_NET_DynAcknowledge(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const void* buffer,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);

TCPIP_HTTP_NET_DynamicWrite Function

Writes a data buffer to the current connection

File

http_net.h

C

```
bool TCPIP_HTTP_NET_DynamicWrite(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const void * buffer, uint16_t
size, bool needAck);
```

Returns

True if the data buffer has been queued for output. False if an invalid buffer was provided or the buffer could not be queued because of the lack of resources (descriptors).

Description

This function takes a buffer and sends it over the HTTP connection as part of the HTTP dynamic variable processing

Remarks

The buffer passed in by the user with this call is queued internally using an available dynamic variable buffer descriptor. That means that the buffer has to be persistent. Once the buffer is processed and sent to output, the dynamicAck callback will be called, to inform the user that the corresponding buffer can be reused/freed.

When multiple connections output their dynamic content concurrently the HTTP may run out of dynamic variable buffer descriptors that are used in queuing the requests and the call may fail. If the call failed, because the buffer could not be queued, it may be retried using by returning TCPIP_HTTP_DYN_PRINT_RES_AGAIN in the TCPIP_HTTP_NET_DynPrint callback.

If sequential write calls are done from within the same TCPIP_HTTP_NET_DynPrint call the HTTP module will try to append the new dynamic data to the existent one.

The number of internal HTTP dynamic variables buffer descriptors is controlled by TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER. It can be obtained at run-time using the TCPIP_HTTP_NET_ConnectionDynamicDescriptors function.

Preconditions

varDcpt - a valid dynamic variable descriptor.

Parameters

Parameters	Description
varDcpt	dynamic variable descriptor as passed in the TCPIP_HTTP_NET_DynPrint function
buffer	The pointer to the persistent buffer to be written to the HTTP connection as part of this dynamic variable callback
size	The number of bytes to be written
needAck	if true, once the buffer is processed internally, TCPIP_HTTP_NET_DynAcknowledge will be called

Function

bool TCPIP_HTTP_NET_DynamicWrite(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,
const void * buffer, uint16_t size, bool needAck);

TCPIP_HTTP_NET_DynamicWriteString Function

Helper for writing a string within a dynamic variable context.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_DynamicWriteString(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const char* str, bool
needAck);
```

Returns

True if the data buffer has been queued for output. False if an invalid buffer was provided or the buffer could not be queued because of the lack of resources (descriptors).

Description

This function takes a 0 terminated ACII string and sends it to the HTTP connection as part of the HTTP dynamic variable processing

Remarks

This is just a helper. The actual function called is still TCPIP_HTTP_NET_DynamicWrite. That means that the supplied string has to be persistent.

See the remarks for TCPIP_HTTP_NET_DynamicWrite.

Preconditions

varDcpt - a valid dynamic variable descriptor.

Parameters

Parameters	Description
varDcpt	dynamic variable descriptor as passed in the TCPIP_HTTP_NET_DynPrint function
str	The string to be written
needAck	if true, once the buffer is processed internally, TCPIP_HTTP_NET_DynAcknowledge will be called

Function

bool TCPIP_HTTP_NET_DynamicWriteString(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,

const char* str, bool needAck);

TCPIP_HTTP_NET_DynPrint Function

Inserts dynamic content into a web page

File

http_net.h

C

```
TCPIP_HTTP_DYN_PRINT_RES TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN. Typically this is used when outputting large amounts of data that cannot fit into one single buffer write operation or when the data is not available all at once.

An alternative (legacy) mechanism for that, is using the connection callbackPos calls: (TCPIP_HTTP_NET_ConnectionCallbackPosGet/TCPIP_HTTP_NET_ConnectionCallbackPosSet). This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. If the connection callbackPos is non-zero, the function will be called again after the current data is processed (equivalent to returning TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN). Once the callback completes, set the callback value back to zero and return TCPIP_HTTP_DYN_PRINT_RES_DONE to resume normal servicing of the request.

If the function does not process this dynamic variable callback (or it wants the default action to be performed), it should return TCPIP_HTTP_DYN_PRINT_RES_DEFAULT. Then the default action for that variable will be performed by the HTTP module.

Note that this is an advanced feature and will be used for processing of HTTP dynamic variable keywords. No implementation currently exists for user added variables and the default action in this case will do nothing.

A TCPIP_HTTP_DYN_PRINT_RES specifying the action to be taken. See the TCPIP_HTTP_DYN_PRINT_RES definition for a list of allowed

actions

Description

Functions in this style are implemented by the application developer. This function generates dynamic content to be inserted into web pages.

The function of this type is called when a dynamic variable is located in a web page. (i.e., ~varname~) The name between the tilde '~' characters is passed as the dynamic variable name.

The function currently supports int32_t and string parameters. The dynamic variable parameters are included in the varDcpt data structure that's passed at the time of the call. For example, the variable "~myArray(2,6)~" will generate the call with: varDcpt.dynName = "myArray"; varDcpt.nArgs = 2; varDcpt.dynArgs->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32; varDcpt.dynArgs->argInt32 = 2; (varDcpt.dynArgs + 1)->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32; (varDcpt.dynArgs + 1)->argInt32 = 6;

When called, this function should write its output to the HTTP connection using the TCPIP_HTTP_NET_DynamicWrite/TCPIP_HTTP_NET_DynamicWriteString functions.

In situations where the dynamic variable print function needs to perform additional write operations, or simply needs to be called again, it must

Remarks

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos or the connection data buffer for storage associated with individual requests.

The varDcpt variable descriptor is valid only in the context of this call. Any parameters that are needed for further processing should be copied to user storage.

The varDcpt variable descriptor is constant and should not be modified in any way by the callback.

If the callback returned TCPIP_HTTP_DYN_PRINT_RES_DEFAULT, no further calls will be made to the callback function.

Returning TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN is an alternative mechanism to manipulation the connection status with TCPIP_HTTP_NET_ConnectionCallbackPosSet/TCPIP_HTTP_NET_ConnectionCallbackPosGet functions (which mechanism is still valid).

- If the TCPIP_HTTP_NET_DynPrint() returned TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN, the HTTP server will start sending the dynamic variable buffer (invoked by TCPIP_HTTP_NET_DynamicWrite/TCPIP_HTTP_NET_DynamicWriteString). Once sending is done, the HTTP server will again call TCPIP_HTTP_NET_DynPrint regardless of the value set with the callback position.
- If the TCPIP_HTTP_NET_DynPrint() returned TCPIP_HTTP_DYN_PRINT_RES_AGAIN, there will be no processing of dynamic variable data (potentially written to the connection) but the TCPIP_HTTP_NET_DynPrint will be invoked again. Returning TCPIP_HTTP_DYN_PRINT_RES_AGAIN is needed when TCPIP_HTTP_NET_DynamicWrite failed or some condition is not satisfied and the TCPIP_HTTP_NET_DynPrint needs to be called again.

The callback position mechanism will be evaluated after the TCPIP_HTTP_NET_DynPrint returns TCPIP_HTTP_DYN_PRINT_RES_DONE.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
varDcpt	pointer to a variable descriptor containing the dynamic variable name and arguments valid within this call context only.
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

```
TCPIP_HTTP_DYN_PRINT_RES TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const                    TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,
const                    TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

TCPIP_HTTP_NET_EventReport Function

Reports an event that occurred in the processing of a HTTP web page

File

http_net.h

C

```
void TCPIP_HTTP_NET_EventReport(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType,
const void* evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. This function reports an event that occurred when processing a file, dynamic variable, etc.

Remarks

The evInfo parameter is valid only in the context of this call. Any data that is needed for further processing should be copied to user storage.

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
evType	the HTTP event that occurred
evInfo	additional info for that particular event • TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR: 0 • TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DEPTH_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED: SSI argument pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN: SSI attribute pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED: SSI variable pointer • TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY: file name pointer • TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED: string with excess number of bytes that cannot be read Valid within this call context only!
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

```
void TCPIP_HTTP_NET_EventReport( TCPIP_HTTP_NET_CONN_HANDLE connHandle,
                                 TCPIP_HTTP_NET_EVENT_TYPE evType, const void* evInfo,
const struct                     _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

TCPIP_HTTP_NET_Task Function

Standard TCP/IP stack module task function.

File

http_net.h

C

```
void TCPIP_HTTP_NET_Task();
```

Returns

None.

Description

This function performs HTTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The HTTP module should have been initialized.

Function

```
void  TCPIP_HTTP_NET_Task(void)
```

TCPIP_HTTP_NET_URLDecode Function

Parses a string from URL encoding to plain text.

File

http_net.h

C

```
uint8_t* TCPIP_HTTP_NET_URLDecode(uint8_t* cData);
```

Returns

A pointer to the last null terminator in data, which is also the first free byte for new data.

Description

This function parses a string from URL encoding to plain text. The following conversions are made: '=' to '0', '&' to '0', '+' to ' ', and "%xx" to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

Remarks

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

Parameters	Description
cData	The string which is to be decoded in place.

Function

uint8_t* TCPIP_HTTP_NET_URLDecode(uint8_t* cData)

TCPIP_HTTP_NET_UserHandlerDeregister Function

Deregisters a previously registered HTTP user handler.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_UserHandlerDeregister(TCPIP_HTTP_NET_USER_HANDLE hHttp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered or there are active connections

Description

This function deregisters a HTTP user callback handler.

Remarks

The call will fail if there is active HTTP traffic. The handler cannot be deregistered while HTTP traffic is in progress.

Preconditions

The HTTP server module properly initialized.

Parameters

Parameters	Description
hHttp	A handle returned by a previous call to TCPIP_HTTP_NET_UserHandlerRegister

Function

TCPIP_HTTP_NET_UserHandlerDeregister(TCPIP_HTTP_NET_USER_HANDLE hHttp)

TCPIP_HTTP_NET_UserHandlerRegister Function

Registers a user callback structure with the HTTP server

File

http_net.h

C

```
TCPIP_HTTP_NET_USER_HANDLE TCPIP_HTTP_NET_UserHandlerRegister(const TCPIP_HTTP_NET_USER_CALLBACK*
userCallback);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

The function registers a user callback data structure with the HTTP server. The HTTP server will call the user supplied callbacks at run-time when processing the web pages.

Remarks

Currently only one user callback structure could be registered. The call will fail if a user callback structure is already registered.

Preconditions

The HTTP server module properly initialized.

Example

Parameters

Parameters	Description
userCallback	user callback to be registered with the HTTP server

Function

TCPIP_HTTP_NET_USER_HANDLE TCPIP_HTTP_NET_UserHandlerRegister
(const TCPIP_HTTP_NET_USER_CALLBACK* userCallback);

TCPIP_HTTP_NET_ConnectionDataBufferSizeGet Function

Returns the size of the connection general purpose data buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Size of the connection's general purpose data buffer.

Description

This function returns the size of the HTTP connection internal data buffer.

Remarks

This is the parameter that was used for HTTP initialization in TCPIP_HTTP_NET_MODULE_CONFIG::dataLen.

Preconditions

None.

Example

```
  // Read a name
  uint8_t* httpDataBuff = TCPIP_HTTP_NET_ConnectionDataBufferGet(connHandle);
  uint16_t httpDataLen = TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(connHandle);
  if(TCPIP_HTTP_NET_ConnectionPostNameRead(connHandle, httpDataBuff, httpDataLen) ==
TCPIP_HTTP_NET_READ_INCOMPLETE)
  {
      return TCPIP_HTTP_NET_IO_RES_NEED_DATA;
  }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionDynamicDescriptors Function

Returns the number of dynamic variable descriptors

File

http_net.h

C

```
int TCPIP_HTTP_NET_ConnectionDynamicDescriptors();
```

Returns

The number of descriptors allocated by the HTTP module.

Description

This function returns the number of the dynamic variable buffer descriptors that are allocated by the HTTP for dynamic variable processing.

Remarks

Currently the dynamic variable descriptors are allocated at the HTTP initialization and cannot be changed at run-time.

Preconditions

None

Function

int TCPIP_HTTP_NET_ConnectionDynamicDescriptors(void);

TCPIP_HTTP_NET_SSINotification Function

Reports an SSI processing event that occurs in the processing of a HTTP web page

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSINotification(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT*
pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

true - if the SSI processing is complete and no further action needs to be taken by the HTTP server false - the HTTP should process the SSI command

Description

Functions in this style are implemented by the application developer. This function is used by the HTTP server to report that an SSI command line was encountered while processing a file. The user has a chance to inspect the line and to provide custom action by using the SSI API functions.

Remarks

The ssiLine parameter is valid only in the context of this call. Any data that is needed for further processing should be copied to user storage.

This function could use the SSI API functions to alter/examine the value of SSI variables that are part of the ssiLine.

This function may NOT write to the network transport buffer. Possiblity to write to the transport channel could be eventually added, based on the dynamic variable model TCPIP_HTTP_NET_DynamicWrite.

The default return value should be false: the user function examines/adjusts value of SSI variables and instructs the HTTP module continue normally. However, the user has the option of supressing the output (SSI "echo") or override the value of a variable (SSI "set"), etc.

Preconditions

SSI processing should be enabled.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pSSINotifyDcpt	pointer to a SSI notification descriptor containing:
fileName	the name of the file the SSI command belongs to
ssiCommand	the SSI command parsed from the command line
nAttribs	number of attribute descriptors in the command
pAttrDcpt	pointer to an array of descriptors parsed from this SSI command
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

bool TCPIP_HTTP_NET_SSINotification(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);

TCPIP_HTTP_NET_SSIVariableDelete Function

Function to delete an SSI variable.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSIVariableDelete(const char* varName);
```

Returns

true - if the variable existed and was deleted false - the variable didn't exist

Description

This function deletes an SSI variable if it exists.

Remarks

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable name

Function

bool TCPIP_HTTP_NET_SSIVariableDelete(const char* varName);

TCPIP_HTTP_NET_SSIVariableGet Function

Function to get access to an existing SSI variable.

File

http_net.h

C

```
const char* TCPIP_HTTP_NET_SSIVariableGet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t*
pVarInt);
```

Returns

a valid pointer to the SSI variable string representation if the variable was found 0 if there is no variable with such name

Description

This function performs a search for the corresponding SSI variable name and returns a pointer to the variable string representation.

Remarks

The returned value points to the internal SSI variable representation. This pointer should not be used for changing the SSI variable value.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable to search for
pVarType	address to store the type of the SSI variable
pVarInt	address to store the integer value of this variable, if the variable exists and is an integer else this address won't be modified Could be NULL if not needed

Function

const char* TCPIP_HTTP_NET_SSIVariableGet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);

TCPIP_HTTP_NET_SSIVariableGetByIndex Function

Function to get access to an existing SSI variable.

File

http_net.h

C

```
const char* TCPIP_HTTP_NET_SSIVariableGetByIndex(int varIndex, const char** pVarName,
TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);
```

Returns

a valid pointer to the SSI variable string representation if the variable was found 0 if there is no variable with such name

Description

This function accesses the corresponding SSI variable by its index and returns a pointer to the variable string representation.

Remarks

The returned value points to the internal SSI variable representation. This pointer should not be used for changing the SSI variable value.

The pVarName points to the internal SSI variable name representation. This pointer MUST NOT be used for changing the SSI variable name.

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varIndex	the index of the SSI variable to search for Should be < TCPIP_HTTP_NET_SSIVariablesNumberGet()
pVarName	address to store a pointer to the variable name
pVarType	address to store the type of the SSI variable
pVarInt	address to store the integer value of this variable, if the variable exists and is an integer else this address won't be modified Could be NULL if not needed

Function

const char* TCPIP_HTTP_NET_SSIVariableGetByIndex(int varIndex, const char** pVarName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);

TCPIP_HTTP_NET_SSIVariableSet Function

Function to set an SSI variable.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSIVariableSet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE varType, const char*
strValue, int32_t intValue);
```

Returns

true - if the variable was updated or created successfully false - the variable didn't exist and the attempt to create it failed (all slots already in use. Increase TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER).

Description

This function sets the new values for an SSI variable. If a variable with such name does not exist, it is created.

Remarks

The string variable interpretation is needed even if the variable is of integer type. HTTP module will use that representation instead of doing an itoa conversion on intValue. value points to the internal SSI variable representation.

The string representation should not exceed TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH. Any excess variables will be truncated.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable name
varType	the type of the SSI variable
strValue	pointer to the string representation of the variable
intValue	the integer value of the variable, if the type is integer.

Function

bool TCPIP_HTTP_NET_SSIVariableSet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE varType, const char* strValue, int32_t intValue);

TCPIP_HTTP_NET_SSIVariablesNumberGet Function

Function to get the number of the current SSI variables.

File

http_net.h

C

```
int TCPIP_HTTP_NET_SSIVariablesNumberGet(int* pMaxNo);
```

Returns

The number of current SSI variables that are in use

Description

This function returns the number of SSI variables that currently exist. It will also return the maximum number of variables that the SSI can hold.

Remarks

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
pMaxNo	pointer to store the maximum number of SSI variables that can exist Can be NULL if not needed

Function

int TCPIP_HTTP_NET_SSIVariablesNumberGet(int* pMaxNo);

TCPIP_HTTP_NET_ConnectionHandleGet Function

Gets the connection handle of a HTTP connection.

File

http_net.h

C

```
TCPIP_HTTP_NET_CONN_HANDLE TCPIP_HTTP_NET_ConnectionHandleGet(int connIx);
```

Returns

A valid connection handle if the connection index is valid 0 if there is no such connection

Description

This function will return the connection handle of the requested HTTP connection index.

Remarks

None

Preconditions

None.

Example

```
TCPIP_HTTP_NET_CONN_HANDLE connHandle;

connHandle = TCPIP_HTTP_NET_ConnectionHandleGet(0);
```

Parameters

Parameters	Description
connIx	the HTTP connection ix.

Function

TCPIP_HTTP_NET_CONN_HANDLE TCPIP_HTTP_NET_ConnectionHandleGet(int connIx);

TCPIP_HTTP_NET_ConnectionIndexGet Function

Gets the index of the HTTP connection.

File

http_net.h

C

```
int TCPIP_HTTP_NET_ConnectionIndexGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The connection index.

Description

This function will return the index of the requested HTTP connection.

Remarks

None

Preconditions

None.

Example

```
int connIx;

connIx = TCPIP_HTTP_NET_ConnectionIndexGet(connHandle);
```

Parameters

Parameters	Description
connHandle	the HTTP connection handle.

Function

int TCPIP_HTTP_NET_ConnectionIndexGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

b) Data Types and Constants

TCPIP_HTTP_DYN_ARG_DCPT Structure

HTTP dynamic argument descriptor.

File

http_net.h

C

```
typedef struct {
  uint16_t argType;
  uint16_t argFlags;
  union {
    int32_t argInt32;
    const char* argStr;
  }
} TCPIP_HTTP_DYN_ARG_DCPT;
```

Members

Members	Description
uint16_t argType;	a TCPIP_HTTP_DYN_ARG_TYPE value
uint16_t argFlags;	extra argument flags
int32_t argInt32;	use this member when the arg type is INT32
const char* argStr;	use this member when the arg type is string

Description

Data structure: TCPIP_HTTP_DYN_ARG_DCPT

This data type defines the structure of a HTTP dynamic variable argument. It is used for describing the dynamic variables arguments.

Remarks

The argument flags are currently not used. They are meant for further extensions.

TCPIP_HTTP_DYN_ARG_TYPE Enumeration

HTTP supported dynamic variables argument types.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_DYN_ARG_TYPE_INVALID = 0,
  TCPIP_HTTP_DYN_ARG_TYPE_INT32,
  TCPIP_HTTP_DYN_ARG_TYPE_STRING
} TCPIP_HTTP_DYN_ARG_TYPE;
```

Members

Members	Description
TCPIP_HTTP_DYN_ARG_TYPE_INVALID = 0	invalid argument type
TCPIP_HTTP_DYN_ARG_TYPE_INT32	the dynamic variable argument is a int32_t
TCPIP_HTTP_DYN_ARG_TYPE_STRING	the dynamic variable argument is a ASCII string

Description

Enumeration: TCPIP_HTTP_DYN_ARG_TYPE

This enumeration defines the types of the HTTP supported dynamic variables arguments.

Remarks

Currently a dynamic variable can have either string or int32_t parameters.

Only 16 bit values are currently supported.

TCPIP_HTTP_DYN_PRINT_RES Enumeration

Dynamic print result when a dynamic variable print callback function returns;

File

http_net.h

C

```
typedef enum {
```

```
  TCPIP_HTTP_DYN_PRINT_RES_DONE = 0,
  TCPIP_HTTP_DYN_PRINT_RES_DEFAULT,
  TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN,
  TCPIP_HTTP_DYN_PRINT_RES_AGAIN
} TCPIP_HTTP_DYN_PRINT_RES;
```

Members

Members	Description
TCPIP_HTTP_DYN_PRINT_RES_DONE = 0	dynamic callback is done
TCPIP_HTTP_DYN_PRINT_RES_DEFAULT	no implementation supported for this dynamic variable, call the default action
TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN	process the action of this call and then call again data written to the HTTP connection but more data is pending
TCPIP_HTTP_DYN_PRINT_RES_AGAIN	do not process any action just call again no data was available to be written to the connection or the written data should be ignored this turn

Description

Enumeration: TCPIP_HTTP_DYN_PRINT_RES

This enumeration defines the results associated with the HTTP dynamic variables callbacks (TCPIP_HTTP_NET_DynPrint).

Remarks

Currently the default action for a user defined dynamic variable is "do nothing". Keywords like "inc" have an internal implementation and don't need to be processed by the user.

TCPIP_HTTP_DYN_VAR_DCPT Structure

HTTP dynamic variable descriptor.

File

http_net.h

C

```
typedef struct {
  const char* dynName;
  const char* fileName;
  uint16_t callbackID;
  uint8_t dynFlags;
  uint8_t nArgs;
  TCPIP_HTTP_DYN_ARG_DCPT* dynArgs;
  const void* dynContext;
} TCPIP_HTTP_DYN_VAR_DCPT;
```

Members

Members	Description
const char* dynName;	ASCII string storing the dynamic variable name
const char* fileName;	ASCII string storing the file name the dynamic variable belongs to
uint16_t callbackID;	Call back ID: the dynamic variable index within the file
uint8_t dynFlags;	a TCPIP_HTTP_DYN_VAR_FLAGS value
uint8_t nArgs;	number of arguments that the variable has
TCPIP_HTTP_DYN_ARG_DCPT* dynArgs;	array of argument descriptors carrying the dynamic variable arguments
const void* dynContext;	dynamic context of the callback. This context is used by the HTTP server and is irrelevant to the user.

Description

Data structure: TCPIP_HTTP_DYN_VAR_DCPT

This data type defines the structure of a HTTP dynamic variable descriptor. When the user registers a TCPIP_HTTP_NET_DynPrint function for dynamic variable processing, this callback will receive the dynamic variable descriptor as a parameter.

Remarks

None.

TCPIP_HTTP_DYN_VAR_FLAGS Enumeration

HTTP supported dynamic variables flags.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_DYN_VAR_FLAG_NONE = 0,
  TCPIP_HTTP_DYN_VAR_FLAG_NAME_TRUNCATED = 0x01,
  TCPIP_HTTP_DYN_VAR_FLAG_ARG_NAME_TRUNCATED = 0x02,
  TCPIP_HTTP_DYN_VAR_FLAG_ARG_NO_TRUNCATED = 0x04
} TCPIP_HTTP_DYN_VAR_FLAGS;
```

Members

Members	Description
TCPIP_HTTP_DYN_VAR_FLAG_NONE = 0	no flag associated with this dynamic variable
TCPIP_HTTP_DYN_VAR_FLAG_NAME_TRUNCATED = 0x01	dynamic variable field exceeded available parsing space the dynamic variable name has been truncated
TCPIP_HTTP_DYN_VAR_FLAG_ARG_NAME_TRUNCATED = 0x02	dynamic variable field exceeded available parsing space the dynamic variable arguments have been truncated
TCPIP_HTTP_DYN_VAR_FLAG_ARG_NO_TRUNCATED = 0x04	dynamic variable arguments exceeded available buffering space the number of arguments of the dynamic variable has been truncated

Description

Enumeration: TCPIP_HTTP_DYN_VAR_FLAGS

This enumeration defines the flags associated with the HTTP supported dynamic variables.

Remarks

Multiple flags con be set simultaneously. New flags will be eventually added. Only 16-bit values are currently supported.

TCPIP_HTTP_NET_CONN_HANDLE Type

File

http_net.h

C

```
typedef const void* TCPIP_HTTP_NET_CONN_HANDLE;
```

Description

HTTP connection identifier, handle of a HTTP connection

TCPIP_HTTP_NET_EVENT_TYPE Enumeration

HTTP reported run-time events.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_EVENT_NONE = 0,
  TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE,
  TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR = -1,
  TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR = -2,
  TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR = -3,
  TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR = -4,
  TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR = -5,
  TCPIP_HTTP_NET_EVENT_FILE_PARSE_ERROR = -6,
  TCPIP_HTTP_NET_EVENT_DEPTH_ERROR = -7,
  TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR = -8,
  TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR = -9,
  TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR = -10,
  TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR = -11,
  TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR = -12,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR = -13,
  TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR = -14,
```

```
  TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR = -15,
  TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR = -16,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR = -17,
  TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR = -18,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED = 0x8001,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED = 0x8002,
  TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED = 0x8003,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED = 0x8004,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN = 0x8005,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH = 0x8006,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED = 0x8007,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN = 0x8008,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID = 0x8009,
  TCPIP_HTTP_NET_EVENT_SSI_HASH_CREATE_FAILED = 0x800a,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED = 0x800b,
  TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY = 0x8020,
  TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY = 0x8021,
  TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED = 0x8030
} TCPIP_HTTP_NET_EVENT_TYPE;
```

Members

Members	Description
TCPIP_HTTP_NET_EVENT_NONE = 0	no event
TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE	Notification that a FS upload operation was completed successfully
TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR = -1	errors an error occurred when opening a HTTP file
TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR = -2	a file name was not specified
TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR = -3	a file name was longer than the HTTP storage space
TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR = -4	file size error
TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR = -5	file read error
TCPIP_HTTP_NET_EVENT_FILE_PARSE_ERROR = -6	file parse error: line too long
TCPIP_HTTP_NET_EVENT_DEPTH_ERROR = -7	the depth allowed for a recursive call was exceeded
TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR = -8	an error occurred when parsing: dynamic variable name terminator could not be found, for example
TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR = -9	a write error was reported while performing an FS upload
TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR = -10	a write error was reported while mounting after an FS upload
TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR = -11	the number of retries for getting a chunk from the pool has been exceeded
TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR = -12	the number of retries for getting a file buffer has been exceeded
TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR = -13	out of memory when trying to allocate space for a dynamic variable descriptor Note that for the dynamic variable descriptor allocation the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR = -14	out of memory when trying to allocate space for a file upload Note that for the file upload buffer allocation the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR = -15	an error occurred when parsing an SSI command: SSI terminator could not be found, for example
TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR = -16	an unknown/unsupported SSI command
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR = -17	a SSI attribute error : command w/o attribute, etc.
TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR = -18	out of memory when trying to allocate space for a SSI command descriptor Note that for the SSI commands the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED = 0x8001	warning: a dynamic variable argument name too long, truncated
TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED = 0x8002	warning: too many arguments for a dynamic variable, truncated

TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED = 0x8003	warning: too many retries for a dynamic variable, stopped
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED = 0x8004	warning: too many attributes for a SSI command, truncated
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN = 0x8005	warning: unrecognized/unsupported SSI command attribute
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH = 0x8006	warning: wrong number of SSI command attributes
TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED = 0x8007	warning: number of SSI set variables exceeded
TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN = 0x8008	warning: SSI variable does not exist
TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID = 0x8009	warning: SSI variable is void: not echoed
TCPIP_HTTP_NET_EVENT_SSI_HASH_CREATE_FAILED = 0x800a	warning: SSI variable hash could not be created, allocation failed Ther ewill be no run time SSI variable support
TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED = 0x800b	event: SSI variable deleted
TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY = 0x8020	warning: allocation from the HTTP chunk pool failed (the allocation will be retried)
TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY = 0x8021	warning: allocation from the HTTP file buffers pool failed (the allocation will be retried)
TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED = 0x8030	warning: the HTTP peek buffer is too small and cannot contain all the data available in the transport socket buffer and a HTTP search operation may fail This will happen only for transport sockets that do not support peek operation with an offset parameter

Description

Enumeration: TCPIP_HTTP_NET_EVENT_TYPE

This enumeration defines the types of the HTTP reported events when processing dynamic variables, files, etc.

Remarks

Multiple warnings can be set simultaneously.

TCPIP_HTTP_NET_IO_RESULT Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_IO_RES_DONE = 0u,
  TCPIP_HTTP_NET_IO_RES_NEED_DATA,
  TCPIP_HTTP_NET_IO_RES_WAITING,
  TCPIP_HTTP_NET_IO_RES_ERROR
} TCPIP_HTTP_NET_IO_RESULT;
```

Members

Members	Description
TCPIP_HTTP_NET_IO_RES_DONE = 0u	Finished with procedure
TCPIP_HTTP_NET_IO_RES_NEED_DATA	More data needed to continue, call again later
TCPIP_HTTP_NET_IO_RES_WAITING	Waiting for asynchronous process to complete, call again later
TCPIP_HTTP_NET_IO_RES_ERROR	Some error has occurred, operation will be aborted

Description

Result states for execution callbacks

TCPIP_HTTP_NET_MODULE_CONFIG Structure

File

http_net.h

C

```
typedef struct {
  uint16_t nConnections;
```

```
  uint16_t dataLen;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
  uint16_t listenPort;
  uint16_t nDescriptors;
  uint16_t nChunks;
  uint16_t maxRecurseLevel;
  uint16_t configFlags;
  uint16_t nFileBuffers;
  uint16_t fileBufferSize;
  uint16_t chunkPoolRetries;
  uint16_t fileBufferRetries;
  uint16_t dynVarRetries;
} TCPIP_HTTP_NET_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous HTTP connections allowed
uint16_t dataLen;	size of the data buffer for reading cookie and GET/POST arguments (bytes)
uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default
uint16_t listenPort;	HTTP listening port: 80, 443, etc.
uint16_t nDescriptors;	how many buffers descriptors for dynamic variable processing to create they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
uint16_t nChunks;	maximum number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL All the chunks are in a pool and are used by all connections
uint16_t maxRecurseLevel;	The maximum depth of recursive calls for serving a web page: • files without dynvars: 1 • file including another file: + 1 • file including a dynamic variable: + 1 etc.
uint16_t configFlags;	a TCPIP_HTTP_NET_MODULE_FLAGS value.
uint16_t nFileBuffers;	number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of maximum files that can be open simultaneously: i.e. for file1 ->include file2 -> include file3 you'll need >= 3 file process buffers
uint16_t fileBufferSize;	size of each of these file buffers should correspond to TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE
uint16_t chunkPoolRetries;	how many retries to get chunk from the pool before giving up
uint16_t fileBufferRetries;	how many retries to get a fileBuffer before giving up
uint16_t dynVarRetries;	how many retries to have for a dynamic variable dynamicPrint function before calling it done

Description

HTTP module dynamic configuration data

TCPIP_HTTP_NET_MODULE_FLAGS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_MODULE_FLAG_DEFAULT = 0x00,
  TCPIP_HTTP_NET_MODULE_FLAG_NON_PERSISTENT = 0x01,
  TCPIP_HTTP_NET_MODULE_FLAG_NO_DELAY = 0x02,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON = 0x10,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF = 0x20,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_DEFAULT = 0x00
```

```
} TCPIP_HTTP_NET_MODULE_FLAGS;
```

Members

Members	Description
TCPIP_HTTP_NET_MODULE_FLAG_DEFAULT = 0x00	Default flags value
TCPIP_HTTP_NET_MODULE_FLAG_NON_PERSISTENT = 0x01	Use non-persistent connections This flag will cause the HTTP connections to be non-persistent and closed after serving each request to the client By default the HTTP connections are persistent
TCPIP_HTTP_NET_MODULE_FLAG_NO_DELAY = 0x02	Create the HTTP sockets with NO-DELAY option. It will flush data as soon as possible.
TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON = 0x10	All HTTP connections have to be secure (supposing the network presentation layer supports encryption) Cannot be used together with TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF
TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF = 0x20	HTTP connections will be non-secure Cannot be used together with TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON
TCPIP_HTTP_NET_MODULE_FLAG_SECURE_DEFAULT = 0x00	HTTP security is based on the port numbers

Description

HTTP module configuration flags Multiple flags can be OR-ed

TCPIP_HTTP_NET_READ_STATUS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_READ_OK = 0,
  TCPIP_HTTP_NET_READ_TRUNCATED,
  TCPIP_HTTP_NET_READ_INCOMPLETE
} TCPIP_HTTP_NET_READ_STATUS;
```

Members

Members	Description
TCPIP_HTTP_NET_READ_OK = 0	Read was successful
TCPIP_HTTP_NET_READ_TRUNCATED	Buffer overflow prevented by truncating value
TCPIP_HTTP_NET_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

Description

Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair

TCPIP_HTTP_NET_STATUS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_STAT_GET = 0u,
  TCPIP_HTTP_NET_STAT_POST,
  TCPIP_HTTP_NET_STAT_BAD_REQUEST,
  TCPIP_HTTP_NET_STAT_UNAUTHORIZED,
  TCPIP_HTTP_NET_STAT_NOT_FOUND,
  TCPIP_HTTP_NET_STAT_OVERFLOW,
  TCPIP_HTTP_NET_STAT_INTERNAL_SERVER_ERROR,
  TCPIP_HTTP_NET_STAT_NOT_IMPLEMENTED,
  TCPIP_HTTP_NET_STAT_REDIRECT,
  TCPIP_HTTP_NET_STAT_TLS_REQUIRED,
  TCPIP_HTTP_NET_STAT_UPLOAD_FORM,
  TCPIP_HTTP_NET_STAT_UPLOAD_STARTED,
  TCPIP_HTTP_NET_STAT_UPLOAD_WRITE,
```

```
  TCPIP_HTTP_NET_STAT_UPLOAD_WRITE_WAIT,
  TCPIP_HTTP_NET_STAT_UPLOAD_OK,
  TCPIP_HTTP_NET_STAT_UPLOAD_ERROR
} TCPIP_HTTP_NET_STATUS;
```

Members

Members	Description
TCPIP_HTTP_NET_STAT_GET = 0u	GET command is being processed
TCPIP_HTTP_NET_STAT_POST	POST command is being processed
TCPIP_HTTP_NET_STAT_BAD_REQUEST	400 Bad Request will be returned
TCPIP_HTTP_NET_STAT_UNAUTHORIZED	401 Unauthorized will be returned
TCPIP_HTTP_NET_STAT_NOT_FOUND	404 Not Found will be returned
TCPIP_HTTP_NET_STAT_OVERFLOW	414 Request-URI Too Long will be returned
TCPIP_HTTP_NET_STAT_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
TCPIP_HTTP_NET_STAT_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
TCPIP_HTTP_NET_STAT_REDIRECT	302 Redirect will be returned
TCPIP_HTTP_NET_STAT_TLS_REQUIRED	403 Forbidden is returned, indicating TLS is required
TCPIP_HTTP_NET_STAT_UPLOAD_FORM	Show the Upload form
TCPIP_HTTP_NET_STAT_UPLOAD_STARTED	An upload operation is being processed
TCPIP_HTTP_NET_STAT_UPLOAD_WRITE	An upload operation is currently writing
TCPIP_HTTP_NET_STAT_UPLOAD_WRITE_WAIT	An upload operation is currently waiting for the write completion
TCPIP_HTTP_NET_STAT_UPLOAD_OK	An Upload was successful
TCPIP_HTTP_NET_STAT_UPLOAD_ERROR	An Upload was not a valid image

Description

Supported Commands and Server Response Codes

TCPIP_HTTP_NET_USER_CALLBACK Structure

HTTP user implemented callback data structure.

File

http_net.h

C

```
typedef struct _tag_TCPIP_HTTP_NET_USER_CALLBACK {
  TCPIP_HTTP_NET_IO_RESULT (* getExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  TCPIP_HTTP_NET_IO_RESULT (* postExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  uint8_t (* fileAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  uint8_t (* userAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser, const char* cPass,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  TCPIP_HTTP_DYN_PRINT_RES (* dynamicPrint)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  void (* dynamicAck)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  void (* eventReport)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType, const void*
evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  bool (* ssiNotify)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
} TCPIP_HTTP_NET_USER_CALLBACK;
```

Members

Members	Description
TCPIP_HTTP_NET_IO_RESULT (* getExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionGetExecute GET process function

TCPIP_HTTP_NET_IO_RESULT (* postExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionPostExecute POST process function
uint8_t (* fileAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionFileAuthenticate File Authenticate function
uint8_t (* userAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser, const char* cPass, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionUserAuthenticate User Authenticate function
TCPIP_HTTP_DYN_PRINT_RES (* dynamicPrint)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_DynPrint Dynamic variable process function
void (* dynamicAck)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_DynAcknowledge Dynamic variable acknowledge function
void (* eventReport)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType, const void* evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_EventReport run-time HTTP processing report
bool (* ssiNotify)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_SSINotification run-time HTTP SSI processing

Description

Structure: TCPIP_HTTP_NET_USER_CALLBACK

This data structure defines the user callbacks that are implemented by the user and the HTTP server calls at run-time.

Remarks

See the detailed explanation for each callback in the callback templates section.

The extra pCBack parameter is passed back for allowing the user to store additional info in the supplied TCPIP_HTTP_NET_USER_CALLBACK data structure that could be used at run-time.

TCPIP_HTTP_NET_USER_HANDLE Type

HTTP user handle.

File

http_net.h

C

```
typedef const void* TCPIP_HTTP_NET_USER_HANDLE;
```

Description

Type: TCPIP_HTTP_NET_USER_HANDLE

A handle that a client can use after the HTTP user callback has been registered.

TCPIP_HTTP_NET_ConnectionPostReadPair Macro

Reads a name and value pair from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
#define TCPIP_HTTP_NET_ConnectionPostReadPair(connHandle, cData, wLen) \
        TCPIP_HTTP_NET_ConnectionPostValueRead(connHandle, cData, wLen)
```

Returns

- TCPIP_HTTP_NET_READ_OK - name and value were successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire name and value could not fit in the buffer, so input was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire name and value was not yet in the buffer, so call this function again later to retrieve

Description

Reads a name and value pair from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates the connection byteCount (see TCPIP_HTTP_NET_ConnectionByteCountGet) by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Remarks

This function is aliased to TCPIP_HTTP_NET_ConnectionPostValueRead, since they effectively perform the same task. The name is provided only for completeness.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostReadPair(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_SSI_ATTR_DCPT Structure

HTTP SSI attribute descriptor.

File

http_net.h

C

```
typedef struct {
  const char* attribute;
  char* value;
} TCPIP_HTTP_SSI_ATTR_DCPT;
```

Members

Members	Description
const char* attribute;	the SSI attribute of the command
char* value;	the SSI value

Description

Data structure: TCPIP_HTTP_SSI_ATTR_DCPT

This data type defines the structure of a SSI command attribute descriptor. When the user registers a TCPIP_HTTP_NET_SSINotification function for SSI processing, this callback will receive info about the SSI attribute descriptors.

Remarks

The SSI commands consist of pairs of (attribute, value) tokens. For example: <!--#set var="varname" value="varvalue" --> has 2 pairs: pair 1 - attribute = var value = varname pair 2 - attribute = value value = varvalue

TCPIP_HTTP_SSI_NOTIFY_DCPT Structure

HTTP SSI notification descriptor.

File

http_net.h

C

```
typedef struct {
  const char* fileName;
  char* ssiCommand;
  int nAttribs;
  TCPIP_HTTP_SSI_ATTR_DCPT* pAttrDcpt;
} TCPIP_HTTP_SSI_NOTIFY_DCPT;
```

Members

Members	Description
const char* fileName;	the file containing the SSI command
char* ssiCommand;	the SSI command parsed from the command line
int nAttribs;	number of attributes descriptors in the command
TCPIP_HTTP_SSI_ATTR_DCPT* pAttrDcpt;	pointer to an array of descriptors parsed from this SSI command

Description

Data structure: TCPIP_HTTP_SSI_NOTIFY_DCPT

This data type defines the structure of a SSI notification descriptor. When the user registers a TCPIP_HTTP_NET_SSINotification function for SSI processing, this callback will receive a SSI descriptor as a parameter.

Remarks

None.

Files

Files

Name	Description
http_net.h	The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server. This module uses the network presentation layer to allow for encrypted connections.
http_net_config.h	HTTP Net configuration file

Description

This section lists the source and header files used by the library.

http_net.h

The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server. This module uses the network presentation layer to allow for encrypted connections.

Enumerations

	Name	Description
	TCPIP_HTTP_DYN_ARG_TYPE	HTTP supported dynamic variables argument types.
	TCPIP_HTTP_DYN_PRINT_RES	Dynamic print result when a dynamic variable print callback function returns;
	TCPIP_HTTP_DYN_VAR_FLAGS	HTTP supported dynamic variables flags.
	TCPIP_HTTP_NET_EVENT_TYPE	HTTP reported run-time events.
	TCPIP_HTTP_NET_IO_RESULT	Result states for execution callbacks
	TCPIP_HTTP_NET_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed

	TCPIP_HTTP_NET_READ_STATUS	Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair
	TCPIP_HTTP_NET_STATUS	Supported Commands and Server Response Codes

Functions

	Name	Description
	TCPIP_HTTP_NET_ActiveConnectionCountGet	Gets the number of active (and inactive) connections.
	TCPIP_HTTP_NET_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_NET_ConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_NET_ConnectionByteCountGet	Returns how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_NET_ConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_NET_ConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDataBufferSizeGet	Returns the size of the connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDiscard	Discards any pending data in the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionDynamicDescriptors	Returns the number of dynamic variable descriptors
	TCPIP_HTTP_NET_ConnectionFileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_NET_ConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_NET_ConnectionFileInclude	Writes a file to the HTTP connection.
	TCPIP_HTTP_NET_ConnectionFlush	Immediately transmits all connection pending TX data.
	TCPIP_HTTP_NET_ConnectionGetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_NET_ConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_NET_ConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionIndexGet	Gets the index of the HTTP connection.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionNetHandle	Returns the network handle of the current connection.
	TCPIP_HTTP_NET_ConnectionPeek	Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.
	TCPIP_HTTP_NET_ConnectionPostExecute	Processes POST form variables and data.
	TCPIP_HTTP_NET_ConnectionPostNameRead	Reads a name from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_NET_ConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_NET_ConnectionPostValueRead	Reads a value from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionRead	Reads an array of data bytes from a connection's RX buffer.
	TCPIP_HTTP_NET_ConnectionReadBufferSize	Returns the size of the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionReadIsReady	Determines how many bytes can be read from the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_NET_ConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_NET_ConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_NET_ConnectionStringFind	Helper to find a string of characters in an incoming connection buffer.
	TCPIP_HTTP_NET_ConnectionUserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_NET_ConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_NET_ConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_NET_DynAcknowledge	Dynamic variable acknowledge function
	TCPIP_HTTP_NET_DynamicWrite	Writes a data buffer to the current connection
	TCPIP_HTTP_NET_DynamicWriteString	Helper for writing a string within a dynamic variable context.
	TCPIP_HTTP_NET_DynPrint	Inserts dynamic content into a web page
	TCPIP_HTTP_NET_EventReport	Reports an event that occurred in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSINotification	Reports an SSI processing event that occurs in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSIVariableDelete	Function to delete an SSI variable.
	TCPIP_HTTP_NET_SSIVariableGet	Function to get access to an existing SSI variable.

	TCPIP_HTTP_NET_SSIVariableGetByIndex	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableSet	Function to set an SSI variable.
	TCPIP_HTTP_NET_SSIVariablesNumberGet	Function to get the number of the current SSI variables.
	TCPIP_HTTP_NET_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_NET_URLDecode	Parses a string from URL encoding to plain text.
	TCPIP_HTTP_NET_UserHandlerDeregister	Deregisters a previously registered HTTP user handler.
	TCPIP_HTTP_NET_UserHandlerRegister	Registers a user callback structure with the HTTP server

Macros

	Name	Description
	TCPIP_HTTP_NET_ConnectionPostReadPair	Reads a name and value pair from a URL encoded string in the network transport buffer.

Structures

	Name	Description
	_tag_TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_DYN_ARG_DCPT	HTTP dynamic argument descriptor.
	TCPIP_HTTP_DYN_VAR_DCPT	HTTP dynamic variable descriptor.
	TCPIP_HTTP_NET_MODULE_CONFIG	HTTP module dynamic configuration data
	TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_SSI_ATTR_DCPT	HTTP SSI attribute descriptor.
	TCPIP_HTTP_SSI_NOTIFY_DCPT	HTTP SSI notification descriptor.

Types

	Name	Description
	TCPIP_HTTP_NET_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	TCPIP_HTTP_NET_USER_HANDLE	HTTP user handle.

Description

HTTP Headers for Microchip TCP/IP Stack

The HTTP module runs a web server within the TCP/IP stack. This facilitates an easy method to view status information and control applications using any standard web browser.

File Name

http_net.h

Company

Microchip Technology Inc.

http_net_config.h

HTTP Net configuration file

Macros

	Name	Description
	TCPIP_HTTP_NET_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_NET_CHUNK_RETRIES	retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted
	TCPIP_HTTP_NET_CHUNKS_NUMBER	number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections
	TCPIP_HTTP_NET_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values

	TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE	Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.
	TCPIP_HTTP_NET_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER	maximum number of arguments for a dynamic variable
	TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER	how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
	TCPIP_HTTP_NET_DYNVAR_MAX_LEN	maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_DYNVAR_PROCESS	This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0
	TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES	retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES	Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE	Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER	Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of... more
	TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_NET_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.
	TCPIP_HTTP_NET_FILENAME_MAX_LEN	maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator
	TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE	Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.
	TCPIP_HTTP_NET_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_NET_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_NET_MAX_HEADER_LEN	The length of longest header string that can be parsed
	TCPIP_HTTP_NET_MAX_RECURSE_LEVEL	The maximum depth of recursive calls for serving a web page: • no dynvars files: 1 • file including a file: 2 • if the include file includes another file: +1 • if a dyn variable: +1 Default value is 3;
	TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE	Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes

	TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER	maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense
	TCPIP_HTTP_NET_SSI_CMD_MAX_LEN	maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE	message to echo when echoing a not found variable
	TCPIP_HTTP_NET_SSI_PROCESS	This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0
	TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER	number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically
	TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH	maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable
	TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH	maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.
	TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER	maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.
	TCPIP_HTTP_NET_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NET_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_NET_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_NET_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_NET_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

Description

HyperText Transfer Protocol (HTTP) Net Configuration file

This file contains the HTTP Net module configuration options

File Name

http_net_config.h

Company

Microchip Technology Inc.

ICMP Module

This section describes the TCP/IP Stack Library ICMP module.

Introduction

TCP/IP Stack Library Internet Control Message Protocol (ICMP) Module for Microchip Microcontrollers

This library provides the API of the ICMP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

Using the Library

This topic describes the basic architecture of the ICMP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `icmp.h`

The interface to the ICMP TCP/IP Stack library is defined in the `icmp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the ICMP TCP/IP Stack library should include `tcpip.h`.

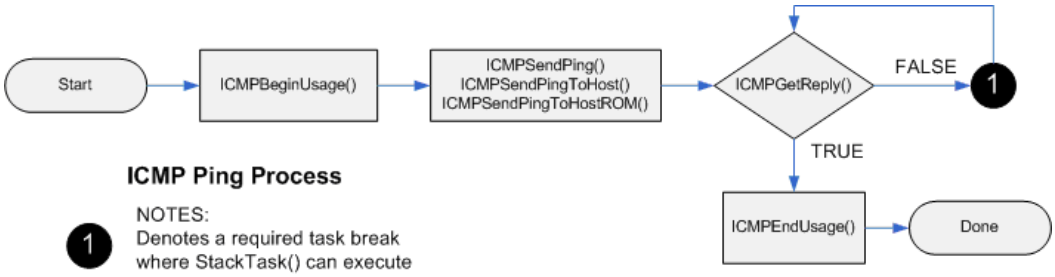
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the ICMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

ICMP 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 ICMP 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

Macros

	Name	Description
	TCPIP_ICMP_CLIENT_USER_NOTIFICATION	allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available
	TCPIP_ICMP_ECHO_REQUEST_TIMEOUT	Timeout for a request that was sent and no reply was received milliseconds
	TCPIP_ICMP_TASK_TICK_RATE	The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the ICMP TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the ICMP TCP/IP Stack. Based on the selections made, the ICMP TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the ICMP TCP/IP Stack.

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.

TCPIP_ICMP_CLIENT_USER_NOTIFICATION Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_CLIENT_USER_NOTIFICATION true
```

Description

allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available

TCPIP_ICMP_ECHO_REQUEST_TIMEOUT Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_ECHO_REQUEST_TIMEOUT 500
```

Description

Timeout for a request that was sent and no reply was received milliseconds

TCPIP_ICMP_TASK_TICK_RATE Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_TASK_TICK_RATE 33
```

Description

The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the ICMP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/icmp.c`	ICMP 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 ICMP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	Deregisters the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.
	TCPIP_ICMP_Task	Standard TCP/IP stack module task function.
	TCPIP_ICMP_EchoRequest	Sends an ICMP echo request containing user supplied data
	TCPIP_ICMP_EchoRequestCancel	Cancels a previously sent ICMP echo request

b) Data Types and Constants

	Name	Description
	ICMP_ECHO_RESULT	result of an ICMP Echo Request call
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered
	TCPIP_ICMP_MODULE_CONFIG	Placeholder for ICMP module configuration.
	_tag_TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST_RESULT	A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)
	TCPIP_ICMP_REQUEST_HANDLE	This is type TCPIP_ICMP_REQUEST_HANDLE.

Description

This section describes the Application Programming Interface (API) functions of the ICMP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_ICMP_CallbackDeregister Function

Deregisters the ICMP callback function.

File

icmp.h

C

```
bool TCPIP_ICMP_CallbackDeregister(ICMP_HANDLE hIcmp);
```

Returns

- true - if the notification handler has been successfully removed
- false - if such a notification handler could not be found

Description

This function notifies a stack client to remove its former registered notification handler. After this operation the client will no longer be notified about the receiving of replies to the ICMP requests.

Remarks

None.

Preconditions

The TCP/IP Stack must be initialized and up and running. A previous successful call to TCPIP_ICMP_CallbackRegister has been done.

Example

```
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);
if(hIcmp != 0)
{
    // successfully registered my handler
    // send requests and process the incoming results
    // ...
    // later on, once we're done, remove the notification handler
    TCPIP_ICMP_CallbackDeregister(hIcmp);
}
```

Parameters

Parameters	Description
hIcmp	an ICMP handle obtained by TCPIP_ICMP_CallbackRegister

Function

bool TCPIP_ICMP_CallbackDeregister(ICMP_HANDLE hIcmp)

TCPIP_ICMP_CallbackRegister Function

Registers a callback to allow the application layer to process incoming ICMPv4 packets

File

icmp.h

C

```
ICMP_HANDLE TCPIP_ICMP_CallbackRegister(void (*callback)(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP,
void * data));
```

Returns

- A non-null handle if the registration succeeded
- 0 if the registration operation failed (for example, out of memory)

Description

Allows a stack client to be notified of the receiving of a response from an ICMP query. Once an Echo request reply is received, the notification handler callback will be called, letting the client know of the result of the query. The callback will contain as parameters:

- the network interface handle on which the query reply was received
- the remote host IP address
- a 32-bit value containing the sequence number in the low 16-bit part and the identifier value in the high 16-bit part.

Remarks

None.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
// Callback function prototype
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);

// ****************************************************************************
// Register callback function
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);
if(hIcmp == 0)
{
    // process error; couldn't register a handler
}

// success; I can send an Echo request and receive notification


// ****************************************************************************
IPV4_ADDR remoteIP = 0xc0a00101;
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;
// send an ICMP query request
TCPIP_ICMP_EchoRequestSend(&remoteIP, mySequenceNumber, myId);


// ****************************************************************************
// process the ICMP reply in the callback function
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data)
{
    // process request from interface hNetIf and remoteIP address
    uint16_t* pReply = (uint16_t*)data;
    uint16_t myRecvId = *pReply;
    uint16_t myRecvSequenceNumber = *(pReply + 1);

    // check that the sequence number, ID and IP address match, etc.
}
```

Parameters

Parameters	Description
callback	notification function to be registered. This function will be called when an echo request reply is received.

Function

ICMP_HANDLE TCPIP_ICMP_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data))

TCPIP_ICMP_EchoRequestSend Function

Sends an ICMP echo request to a remote node.

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV4_ADDR * targetAddr, uint16_t
sequenceNumber, uint16_t identifier);
```

Returns

- ICMP_ECHO_OK - Indicates the query request was successfully sent
- ICMP_ECHO_RESULT - The query request was unsuccessfully sent, which results in an error code (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMP query message to a remote host. The supplied sequence number and identifier will be used in the query message. The user will be notified by the result of the query using a notification handle registered by using the TCPIP_ICMP_CallbackRegister function.

Remarks

This function does not allow the user to specify a data buffer to be sent as part of the request. The preferred function to use is TCPIP_ICMP_EchoRequest.

This function will eventually be obsoleted.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
IPV4_ADDR remoteAddress = 0xc0a00101;
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;

if(TCPIP_ICMP_EchoRequestSend(0, &remoteAddress, mySequenceNumber, myId) == ICMP_ECHO_OK )
{
    // successfully sent the ICMP request
    //
}
else
{
    // process the error
}
```

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestSend (TCPIP_NET_HANDLE netH, IPV4_ADDR * targetAddr, uint16_t sequenceNumber, uint16_t identifier)

TCPIP_ICMP_Task Function

Standard TCP/IP stack module task function.

File

icmp.h

C

```
void TCPIP_ICMP_Task();
```

Returns

None.

Description

This function performs ICMPv4 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ICMP module should have been initialized.

Function

void TCPIP_ICMP_Task(void)

TCPIP_ICMP_EchoRequest Function

Sends an ICMP echo request containing user supplied data

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequest(TCPIP_ICMP_ECHO_REQUEST* pEchoRequest, TCPIP_ICMP_REQUEST_HANDLE*
pHandle);
```

Returns

- ICMP_ECHO_OK - Indicates the query request was successfully sent
- ICMP_ECHO_RESULT - The query request was unsuccessfully sent, which results in an error code (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMP query message to a remote host. The supplied sequence number and identifier will be used in the query message. The request will also contain user supplied data. The request is scheduled and the user will be notified of the outcome of the query using the callback function that's specified in the call.

Remarks

The data buffer that's passed as part of the callback routine is no longer available after the callback routine returns control.

This routine is more generic than TCPIP_ICMP_EchoRequestSend and is preferred.

The user has to enforce the fact that the "identifier" field has to be unique per (destination address, source address) pair.

Currently there could be only one ping/echo request active at a time. If another echo request is active, a ICMP_ECHO_BUSY code will be returned.

Once the callback notification occurs, the echo request is completed and the icmpHandle is invalidated.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
  uint8_t  myDataBuffer[200];     // buffer for the ping data
  void EchoCallback(TCPIP_ICMP_ECHO_REQUEST* pReqData, TCPIP_ICMP_REQUEST_HANDLE icmpHandle,
TCPIP_ICMP_ECHO_REQUEST_RESULT result);    // callback function to be called

  TCPIP_ICMP_ECHO_REQUEST myEchoRequest;
  myEchoRequest.netH  = 0;
  myEchoRequest.targetAddr  = 0xc0a00101;
  myEchoRequest.sequenceNumber = 1;
  myEchoRequest.identifier     = 0x1234;
  myEchoRequest.pData          = myDataBuffer;
  myEchoRequest.dataSize       = sizeof(myDataBuffer);
  myEchoRequest.callback       = EchoCallback;


  if(TCPIP_ICMP_EchoRequest(&myEchoRequest, 0) == ICMP_ECHO_OK )
  {
      // successfully sent the ICMP request
      //
      // EchoCallback() will be called and data can be examined
  }
  else
  {
      // process the error
  }
```

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequest (TCPIP_ICMP_ECHO_REQUEST* pEchoRequest, TCPIP_ICMP_REQUEST_HANDLE* pHandle);

TCPIP_ICMP_EchoRequestCancel Function

Cancels a previously sent ICMP echo request

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestCancel(TCPIP_ICMP_REQUEST_HANDLE icmpHandle);
```

Returns

- ICMP_ECHO_OK - Indicates the cancel request was successful
- ICMP_ECHO_BAD_HANDLE - No such request is currently scheduled Invalid handle

Description

This function allows a stack client to cancel a pending ICMP echo request. The request should have been previously scheduled with TCPIP_ICMP_EchoRequest.

Remarks

None

Preconditions

The TCP/IP Stack must be initialized and up and running. Valid handle obtained using TCPIP_ICMP_EchoRequest.

Example

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestCancel (TCPIP_ICMP_REQUEST_HANDLE icmpHandle);

b) Data Types and Constants

ICMP_ECHO_RESULT Enumeration

File

icmp.h

C

```
typedef enum {
  ICMP_ECHO_OK = 0,
  ICMP_ECHO_ALLOC_ERROR = -1,
  ICMP_ECHO_ROUTE_ERROR = -2,
  ICMP_ECHO_TRANSMIT_ERROR = -3,
  ICMP_ECHO_PARAMETER_ERROR = -4,
  ICMP_ECHO_BUSY = -5,
  ICMP_ECHO_BAD_HANDLE = -6
} ICMP_ECHO_RESULT;
```

Members

Members	Description
ICMP_ECHO_OK = 0	operation successful error codes, < 0
ICMP_ECHO_ALLOC_ERROR = -1	could not allocate memory
ICMP_ECHO_ROUTE_ERROR = -2	could not find a route to destination
ICMP_ECHO_TRANSMIT_ERROR = -3	could not transmit (dead interface, etc.)
ICMP_ECHO_PARAMETER_ERROR = -4	null pointer, etc.
ICMP_ECHO_BUSY = -5	module currently busy
ICMP_ECHO_BAD_HANDLE = -6	invalid handle, no such request exist

Description

result of an ICMP Echo Request call

ICMP_HANDLE Type

File

icmp.h

C

```
typedef const void* ICMP_HANDLE;
```

Description

a handle that a client can use after the event handler has been registered

TCPIP_ICMP_MODULE_CONFIG Structure

Placeholder for ICMP module configuration.

File

icmp.h

C

```
typedef struct {
} TCPIP_ICMP_MODULE_CONFIG;
```

Description

ICMP Module Configuration Structure Typedef

Provides a placeholder for ICMP module configuration.

Remarks

None.

TCPIP_ICMP_ECHO_REQUEST Structure

Contains the data needed for an ICMP request operation

File

icmp.h

C

```
typedef struct _tag_TCPIP_ICMP_ECHO_REQUEST {
  TCPIP_NET_HANDLE netH;
  IPV4_ADDR targetAddr;
  uint16_t sequenceNumber;
  uint16_t identifier;
  uint8_t* pData;
  uint16_t dataSize;
  void (* callback)(const struct _tag_TCPIP_ICMP_ECHO_REQUEST* pReqData, TCPIP_ICMP_REQUEST_HANDLE
icmpHandle, TCPIP_ICMP_ECHO_REQUEST_RESULT result);
} TCPIP_ICMP_ECHO_REQUEST;
```

Description

ICMP Request Data Structure

This structure defines the information required for sending an ICMP request that contains user data.

Remarks

When the callback is made with an error result, the TCPIP_ICMP_ECHO_REQUEST structure may identify the request, if no reply was received (for example timeout)

TCPIP_ICMP_ECHO_REQUEST_RESULT Enumeration

File

icmp.h

C

```
typedef enum {
  TCPIP_ICMP_ECHO_REQUEST_RES_OK = 0,
  TCPIP_ICMP_ECHO_REQUEST_RES_TMO = -1
} TCPIP_ICMP_ECHO_REQUEST_RESULT;
```

Members

Members	Description
TCPIP_ICMP_ECHO_REQUEST_RES_OK = 0	a reply has been successfully received error codes, < 0
TCPIP_ICMP_ECHO_REQUEST_RES_TMO = -1	no reply received; the current request timed out and will be deleted

Description

A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)

TCPIP_ICMP_REQUEST_HANDLE Type

File

icmp.h

C

```
typedef const void* TCPIP_ICMP_REQUEST_HANDLE;
```

Description

This is type TCPIP_ICMP_REQUEST_HANDLE.

Files

Files

Name	Description
icmp.h	The Internet Control Message Protocol is used to send error and status messages and requests.
icmp_config.h	ICMP configuration file

Description

This section lists the source and header files used by the library.

icmp.h

The Internet Control Message Protocol is used to send error and status messages and requests.

Enumerations

	Name	Description
	ICMP_ECHO_RESULT	result of an ICMP Echo Request call
	TCPIP_ICMP_ECHO_REQUEST_RESULT	A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)

Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	Deregisters the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequest	Sends an ICMP echo request containing user supplied data
	TCPIP_ICMP_EchoRequestCancel	Cancels a previously sent ICMP echo request

	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.
	TCPIP_ICMP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	_tag_TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_MODULE_CONFIG	Placeholder for ICMP module configuration.

Types

	Name	Description
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered
	TCPIP_ICMP_REQUEST_HANDLE	This is type TCPIP_ICMP_REQUEST_HANDLE.

Description

ICMP Module Definitions for the Microchip TCP/IP Stack

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

File Name

icmp.h

Company

Microchip Technology Inc.

icmp_config.h

ICMP configuration file

Macros

	Name	Description
	TCPIP_ICMP_CLIENT_USER_NOTIFICATION	allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available
	TCPIP_ICMP_ECHO_REQUEST_TIMEOUT	Timeout for a request that was sent and no reply was received milliseconds
	TCPIP_ICMP_TASK_TICK_RATE	The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Internet Control Message Protocol (ICMP) Configuration file

This file contains the ICMP module configuration options

File Name

icmp_config.h

Company

Microchip Technology Inc.

ICMPv6 Module

This section describes the TCP/IP Stack Library ICMPv6 module.

Introduction

TCP/IP Library ICMPv6 Module for Microchip Microcontrollers

This library provides the API of the ICMPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This document specifies a set of APIs of Internet Control Message Protocol (ICMP) messages for use with version 6 of the Internet Protocol (IPv6).

The Internet Protocol, version 6 (IPv6) is a new version of IP, uses the Internet Control Message Protocol (ICMP) as defined for IPv4

[RFC-792], with a number of changes. It is called ICMPv6, and has an IPv6 Next Header value of 58.

ICMPv6 messages are used by IPv6 nodes to report *error messages* and *information messages*. ICMPv6 is also used for Ipv6 node diagnostic (i.e., IPv6 ping).

The ICMPv6 protocol also provides a framework for the following:

Neighbor Discovery

Neighbor Discovery is a series of five ICMPv6 messages that manage node-to-node communication on a link. Neighbor Discovery replaces Address Resolution Protocol (ARP), ICMP (IPv4) Router Discovery, and the ICMP (IPv4) Redirect message.

Multicast Listener Discovery (MLD)

Multicast Listener Discovery is a series of three ICMP messages that manage subnet multicast membership. Multicast Listener Discovery replaces version 2 of the Internet Group Management Protocol (IGMP) for IPv4.

Path MTU Discovery

The maximum transmission unit (MTU) for a path is the minimum link MTU of all links on a path between a source and a destination. IPv6 packets that are smaller than the path MTU do not require fragmentation by the host and are successfully forwarded by all routers on the path. To discover the path MTU, the sending host uses the receipt of **ICMPv6 Packet Too Big messages**.

ICMPv6 Common Messages	Functions
Echo Request	Sent to check IPv6 connectivity to a particular host.
Echo Reply	Sent in response to an ICMPv6 Echo Request.
Destination Unreachable	Sent by a router or the destination host to inform the sending host that the packet or payload cannot be delivered.
Packet to big	Sent by a router to inform a sending host that a packet is too large to forward.
Time exceeded	Sent by a router to inform a sending host that the Hop Limit of an IPv6 packet has expired.
Parameter Problem	Sent by a router to inform a sending host that an error was encountered in processing the IPv6 header or an IPv6 extension header.

Using the Library

This topic describes the basic architecture of the ICMPv6 TCP/IP Library and provides information and examples on its use.

Description

Interface Header File: `icmpv6.h`

The interface to the ICMPv6 TCP/IP library is defined in the `icmpv6.h` header file. Any C language source (`.c`) file that uses the ICMPv6 TCP/IP library should include `ipv6.h` , `icmpv6.h`, and `ndp.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

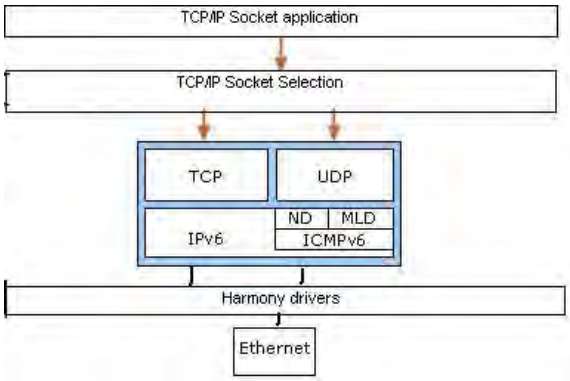
This library provides the API of the ICMPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

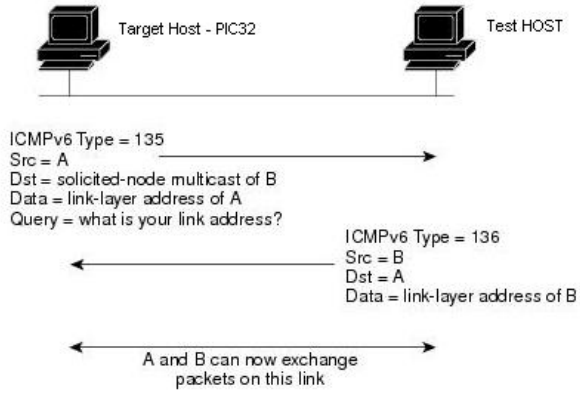
ICMPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



ICMPv6 message communication



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6 and ICMPv6 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

The configuration of the ICMPv6 TCP/IP Stack is based on the file `icmpv6_config.h`.

This header file contains the configuration selection for the ICMPv6 TCP/IP Stack. Based on the selections made, the ICMPv6 TCP/IP Stack IP may support the selected features. These configuration settings will apply to all instances of the ICMPv6 TCP/IP Stack.

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.

Building the Library

This section lists the files that are available in the ICMPv6 module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/icmpv6.c`	ICMPv6 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 ICMPv6 module depends on the following modules:

- TCP/IP Stack Library
- IPv6 Module

Library Interface

a) Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_Flush	Flushes a ICMPv6 packet.
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.
	TCPIP_ICMPV6_EchoRequestSend	Sends an ICMPv6 echo request to a remote node.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.

b) Data Types and Constants

	Name	Description
	ICMPV6_ERR_PTB_CODE	ICMPv6 Packet Too Big error code.
	ICMPV6_INFO_EREQ_CODE	ICMPv6 Packet Echo Request information code.
	ICMPV6_INFO_ERPL_CODE	ICMPv6 Packet Echo Reply information code.
	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.
	ICMPV6_ERR_DU_CODE	ICMPv6 Destination Unreachable error code.
	ICMPV6_ERR_PP_CODE	ICMPv6 Parameter Problem error code.
	ICMPV6_ERR_TE_CODE	ICMPv6 Time Exceeded error code.
	ICMPV6_HANDLE	ICMPv6 handle.
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet.
	ICMPV6_PACKET_TYPES	ICMPv6 packet types.
	TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION	allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Description

This section describes the Application Programming Interface (API) functions of the ICMPv6 module.
Refer to each section for a detailed description.

a) Functions

TCPIP_ICMPV6_CallbackDeregister Function

Deregisters an upper-layer function from ICMPv6 callback.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_CallbackDeregister(ICMPV6_HANDLE hIcmpv6);
```

Returns

- true - On Success
- false - On Failure (If no such handler registered)

Description

This function deregisters a previously registered function from the ICMPv6 register list.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
hIcmpv6	ICMPv6 handler

Function

ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf, uint8_t type, IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, void * header))

TCPIP_ICMPV6_Flush Function

Flushes a ICMPv6 packet.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_Flush(IPV6_PACKET * pkt);
```

Returns

- true - if the packet was flushed
- false - if the packet was queued

Description

Transmit out a ICMPv6 packet.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
pkt	The packet to flush

Function

bool TCPIP_ICMPV6_Flush (IPV6_PACKET * pkt)

TCPIP_ICMPV6_HeaderEchoRequestPut Function

Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

File

icmpv6.h

C

```
IPV6_PACKET * TCPIP_ICMPV6_HeaderEchoRequestPut(TCPIP_NET_HANDLE hNetIf, const IPV6_ADDR * localIP, const
IPV6_ADDR * remoteIP, uint8_t type, uint16_t identifier, uint16_t sequenceNumber);
```

Returns

IPV6_PACKET * - The constructed error packet or NULL

Description

This function allocates a packet using TCPIP_ICMPV6_Open() and updates local and remote addresses. Updates IPv6 Header with ICMPv6 type and upper-layer header for an ICMPv6 echo request.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
pNetIf	The interface for the outgoing packet.
localIP	The local address that should be used for this packet.
remoteIP	The packet's destination address
type	Echo Request or Echo Reply
identifier	The Echo Request id.
sequenceNumber	The Echo request sequence number

Function

IPV6_PACKET * TCPIP_ICMPV6_HeaderEchoRequestPut (TCPIP_NET_HANDLE hNetIf,
IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, uint8_t type,
uint16_t identifier, uint16_t sequenceNumber)

TCPIP_ICMPV6_EchoRequestSend Function

Sends an ICMPv6 echo request to a remote node.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV6_ADDR * targetAddr, uint16_t sequenceNumber,
uint16_t identifier, uint32_t packetSize);
```

Returns

- true - if the query request was successfully sent
- false - otherwise (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMPv6 Echo request message to a remote host. The supplied sequence number and identifier will be used in the query message. The user will be notified by the result of the query using a notification handle registered by using the TCPIP_ICMPV6_CallbackRegister function.

Remarks

None.

Preconditions

The TCP/IP Stack is initialized and up and running. IPv6 and ICMPv6 are initialized.

Function

bool TCPIP_ICMPV6_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV6_ADDR * targetAddr,
uint16_t sequenceNumber, uint16_t identifier,uint32_t packetSize);

TCPIP_ICMPV6_CallbackRegister Function

Registers an upper-layer function for ICMPv6 callback.

File

icmpv6.h

C

```
ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister(void (*callback)(TCPIP_NET_HANDLE hNetIf, uint8_t type, const
IPV6_ADDR * localIP, const IPV6_ADDR * remoteIP, void * header));
```

Returns

ICMPV6_HANDLE

- Success - Returns a valid handle
- Failure - Null handle

Description

This function registers an upper-layer function to handle ICMPv6 messages that may require action at the application layer (Echo Replies, Error messages)

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
type	ICMPv6 header type
localIP	IPv6 destination address of the incoming message
remoteIP	IPv6 address of the node that originated the incoming message
header	Pointer to the ICMPv6 header

Function

ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf,
uint8_t type, IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, void * header))

b) Data Types and Constants

ICMPV6_ERR_PTB_CODE Macro

ICMPv6 Packet Too Big error code.

File

icmpv6.h

C

```
#define ICMPV6_ERR_PTB_CODE 0u
```

Description

Macro: ICMPV6_ERR_PTB_CODE

ICMPv6 packet too big error code value is 0 generated by the originator and ignored by the receiver.

Remarks

A Packet Too Big MUST be sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU of the outgoing link. The information in this message is used as part of the Path MTU Discovery process [PMTU].

More details on RFC - 4446.

ICMPV6_INFO_EREQ_CODE Macro

ICMPv6 Packet Echo Request information code.

File

icmpv6.h

C

```
#define ICMPV6_INFO_EREQ_CODE 0u
```

Description

Macro: ICMPV6_INFO_EREQ_CODE

Definition for ICMPv6 Packet Echo Request information code.

Remarks

More details on RFC - 4446.

ICMPV6_INFO_ERPL_CODE Macro

ICMPv6 Packet Echo Reply information code.

File

icmpv6.h

C

```
#define ICMPV6_INFO_ERPL_CODE 0u
```

Description

Macro: ICMPV6_INFO_ERPL_CODE

Definition for ICMPv6 Packet Echo Reply information code.

Remarks

More details on RFC - 4446.

TCPIP_ICMPV6_PutHeaderEchoReply Macro

File

icmpv6.h

C

```
#define TCPIP_ICMPV6_PutHeaderEchoReply TCPIP_ICMPV6_HeaderEchoRequestPut
```

Description

This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

ICMPV6_ERR_DU_CODE Enumeration

ICMPv6 Destination Unreachable error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_DU_NO_ROUTE = 0u,
  ICMPV6_ERR_DU_PROHIBITED = 1u,
  ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u,
  ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u,
  ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u,
  ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u,
  ICMPV6_ERR_DU_REJECT_ROUTE = 6u
} ICMPV6_ERR_DU_CODE;
```

Members

Members	Description
ICMPV6_ERR_DU_NO_ROUTE = 0u	No route to destination
ICMPV6_ERR_DU_PROHIBITED = 1u	Communication with destination administratively prohibited
ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u	Beyond scope of source address
ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u	Address unreachable
ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u	Port unreachable
ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u	Source address failed ingress/egress policy
ICMPV6_ERR_DU_REJECT_ROUTE = 6u	Reject route to destination

Description

Enumeration: ICMPV6_ERR_DU_CODE

Defines different type of ICMPv6 Destination Unreachable error code.

Remarks

A Destination Unreachable message should be generated by a router, or by the IPv6 layer in the originating node, in response to a packet that cannot be delivered to its destination address for reasons other than congestion. An ICMPv6 message MUST NOT be generated if a packet is dropped due to congestion.

More details on RFC - 4446.

ICMPV6_ERR_PP_CODE Enumeration

ICMPv6 Parameter Problem error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u,
  ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u,
  ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u
} ICMPV6_ERR_PP_CODE;
```

Members

Members	Description
ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u	Erroneous header field encountered
ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u	Unrecognized Next Header type encountered
ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u	Unrecognized IPv6 option encountered

Description

Enumeration: ICMPV6_ERR_PP_CODE

Definitions for ICMPv6 Parameter Problem error code.

Remarks

More details on RFC - 4446.

ICMPV6_ERR_TE_CODE Enumeration

ICMPv6 Time Exceeded error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u,
  ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u
} ICMPV6_ERR_TE_CODE;
```

Members

Members	Description
ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u	Hop limit exceeded in transit
ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u	Fragment reassembly time exceeded

Description

Enumeration: ICMPV6_ERR_TE_CODE

Definitions for ICMPv6 Time Exceeded error code. An ICMPv6 Time Exceeded message with Code 1 is used to report fragment reassembly time out.

Remarks

More details on RFC - 4446.

ICMPV6_HANDLE Type

ICMPv6 handle.

File

icmpv6.h

C

```
typedef const void* ICMPV6_HANDLE;
```

Description

Type: ICMPV6_HANDLE

A handle that a client needs to use when deregistering a notification handler.

Remarks

This handle can be used by the client after the event handler has been registered.

ICMPV6_HEADER_ECHO Structure

File

icmpv6.h

C

```
typedef struct {
  uint8_t vType;
  uint8_t vCode;
```

```
  uint16_t wChecksum;
  uint16_t identifier;
  uint16_t sequenceNumber;
} ICMPV6_HEADER_ECHO;
```

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	Error code
uint16_t wChecksum;	Packet TX and RX checksum
uint16_t identifier;	incoming and outgoing packet identifier
uint16_t sequenceNumber;	request and reply sequence number

Description

Header for an ICMPv6 Echo Request/Reply packet

ICMPV6_HEADER_ERROR Structure

Header for an ICMPv6 Error packet.

File

icmpv6.h

C

```
typedef struct {
  uint8_t vType;
  uint8_t vCode;
  uint16_t wChecksum;
  uint32_t additionalData;
} ICMPV6_HEADER_ERROR;
```

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	error code
uint16_t wChecksum;	Packet TX and RX checksum
uint32_t additionalData;	Unused for Destination. Unreachable and Time Exceeded. MTU for MTU. Pointer for Parameter Problem.

Description

Structure: ICMPV6_HEADER_ERROR

ICMPv6 different code and type error messages (Range 0 to 127).

Remarks

More details on RFC - 4446.

ICMPV6_PACKET_TYPES Enumeration

ICMPv6 packet types.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERROR_DEST_UNREACHABLE = 1u,
  ICMPV6_ERROR_PACKET_TOO_BIG = 2u,
  ICMPV6_ERROR_TIME_EXCEEDED = 3u,
  ICMPV6_ERROR_PARAMETER_PROBLEM = 4u,
  ICMPV6_INFO_ECHO_REQUEST = 128u,
  ICMPV6_INFO_ECHO_REPLY = 129u,
  ICMPV6_INFO_ROUTER_SOLICITATION = 133u,
  ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u,
```

```
  ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u,
  ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u,
  ICMPV6_INFO_REDIRECT = 137u
} ICMPV6_PACKET_TYPES;
```

Members

Members	Description
ICMPV6_ERROR_DEST_UNREACHABLE = 1u	Destination Unreachable error packet
ICMPV6_ERROR_PACKET_TOO_BIG = 2u	Packet Too Big error packet
ICMPV6_ERROR_TIME_EXCEEDED = 3u	Time Exceeded error packet
ICMPV6_ERROR_PARAMETER_PROBLEM = 4u	Parameter Problem error packet
ICMPV6_INFO_ECHO_REQUEST = 128u	Echo Request packet
ICMPV6_INFO_ECHO_REPLY = 129u	Echo Reply packet
ICMPV6_INFO_ROUTER_SOLICITATION = 133u	Router solicitation NDP packet
ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u	Router advertisement NDP packet
ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u	Neighbor Solicitation NDP packet
ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u	Neighbor Advertisement NDP packet
ICMPV6_INFO_REDIRECT = 137u	Redirect NDP packet

Description

Enumeration: ICMPV6_PACKET_TYPES

Defines different type of ICMPv6 message. Values in the range from 0 to 127 indicate an error type of messages. Values in the range 128 to 255 indicate an information message.

Remarks

None.

TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION Macro

File

icmpv6_config.h

C

```
#define TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION true
```

Description

allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Files

Files

Name	Description
icmpv6.h	IPv6 Internet Communication Message Protocol
icmpv6_config.h	ICMPv6 configuration file

Description

This section lists the source and header files used by the library.

icmpv6.h

IPv6 Internet Communication Message Protocol

Enumerations

	Name	Description
	ICMPV6_ERR_DU_CODE	ICMPv6 Destination Unreachable error code.
	ICMPV6_ERR_PP_CODE	ICMPv6 Parameter Problem error code.
	ICMPV6_ERR_TE_CODE	ICMPv6 Time Exceeded error code.
	ICMPV6_PACKET_TYPES	ICMPv6 packet types.

Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.
	TCPIP_ICMPV6_EchoRequestSend	Sends an ICMPv6 echo request to a remote node.
	TCPIP_ICMPV6_Flush	Flushes a ICMPv6 packet.
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

Macros

	Name	Description
	ICMPV6_ERR_PTB_CODE	ICMPv6 Packet Too Big error code.
	ICMPV6_INFO_EREQ_CODE	ICMPv6 Packet Echo Request information code.
	ICMPV6_INFO_ERPL_CODE	ICMPv6 Packet Echo Reply information code.
	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

Structures

	Name	Description
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet.

Types

	Name	Description
	ICMPV6_HANDLE	ICMPv6 handle.

Description

Internet Control Message Protocol (ICMP) IPv6 API Header File

Internet Control Message Protocol (ICMP) in IPv6 functions the same as ICMP in IPv4.ICMPv6 is used to report error messages and information messages for IPv6 nodes. ICMP packets in IPv6 are used in the IPv6 neighbor discovery process, path MTU discovery, and the Multicast Listener Discovery (MLD) protocol for IPv6. RFC - 4443.

File Name

icmpv6.h

Company

Microchip Technology Inc.

icmpv6_config.h

ICMPv6 configuration file

Macros

	Name	Description
	TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION	allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Description

Internet Control Message Protocol for IPv6 (ICMPv6) Configuration file

This file contains the ICMPv6 module configuration options

File Name

icmpv6_config.h

Company

Microchip Technology Inc.

Iperf Module

This section describes the TCP/IP Stack Library Iperf Module.

Introduction

This library provides the API of the Iperf module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Iperf is a networking tool that helps to measure networking bandwidth and performance.

Using the Library

This topic describes the basic architecture of the Iperf TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `iperf.h`

The interface to the Iperf TCP/IP Stack library is defined in the `iperf.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Iperf TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Using Iperf

Describes how to use in the TCP/IP Stack Library Iperf module.

Description

Iperf is a networking tool that helps to measure networking bandwidth and performance. The Iperf module can act as both a client and server for testing. Iperf has the ability to test both UDP and TCP. In the case of UDP, you can specify the size of the UDP datagrams. For TCP, Iperf measures the throughput of the payload.

To run Iperf, you will need a personal computer that has an Iperf application installed. There is an open source version that is maintained, as well as many other variants across the Internet.

Iperf is meant to be run on the command line. On the PIC32 MPLAB Harmony TCP/IP Stack device side, there is also a built-in command console that can accept your different Iperf settings from a system console, such as the UART.

Iperf measures performance data in a unidirectional format. Therefore, the side that the server is running on is considered the receiver, and provides the most accurate performance values.

Command Synopsis

```
iperf [ -s|-c <IP addr> ]
[ -u ]
[ -i <sec> ]
[ -b <bandwidth> ]
[ -t <time> ]
```

`-s`	Runs the Iperf server. No IP address needs to be specified.
`-c <IP addr>`	Runs the Iperf client. The IP address is the IP address of the server.
`-u`	Sends UDP datagrams.
`-i <sec>`	Specified the time interval, in seconds, that the display will be updated.
`-b <bandwidth>`	Specifies the amount of data to try and send. This option is only valid with UDP datagrams.
`-t <time>`	Amount of time, in seconds, to run the test.

Note: The socket size of TCP/UDP (especially TCP) will affect the benchmark result. In addition, activated TCP/IP modules will also consume CPU and Ethernet load. The traffic load in your test network environment will also affect the benchmark test.

To get a reasonable maximum benchmark/evaluation data for a PIC32 device target, it is recommended to disable other modules, such as HTTP, client example, server example, etc., and find a uncongested network.

Running the Demonstration

1. First, make sure the Iperf module and the System Console are enabled in your project.
2. Then, rebuild and program your project into target board.
3. After powering on the development board and associating the connected network, start the server side Iperf application first. If you start Iperf as a server on the development board in the console, this implies that you are trying to measure the PIC32 device Ethernet receiver performance. If you start the Iperf server on a personal computer, you will be measuring PIC Ethernet transmit performance.

PIC32 Ethernet Starter Kit Example

The following tests show receiver and transmitter performance, respectively on the PIC32 Ethernet Starter Kit with the following settings

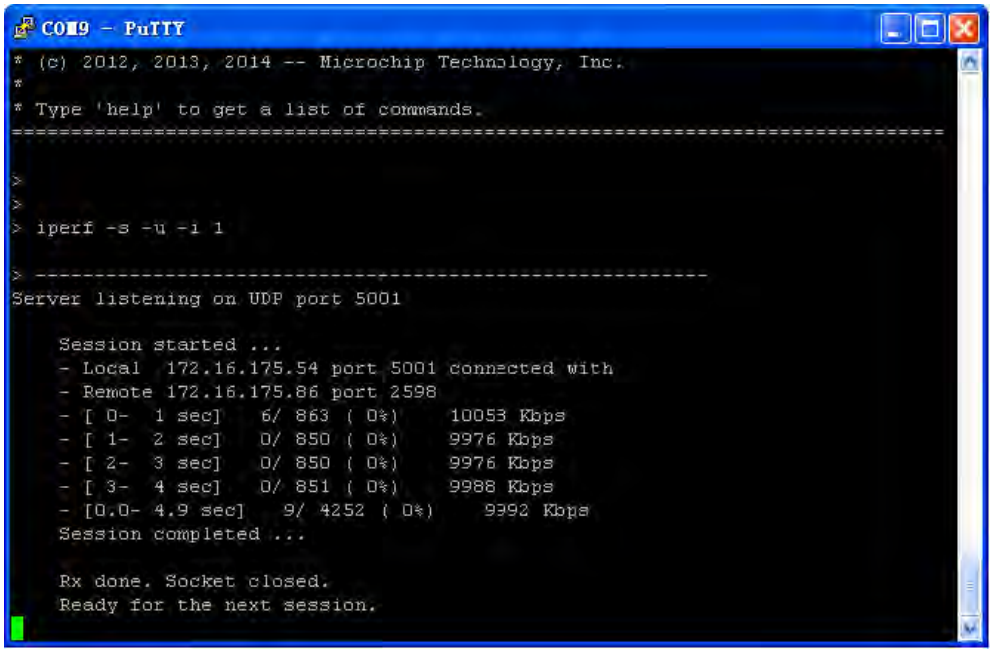
Note: The data does NOT show the maximum throughput of the PIC32 device as the settings are not optimized for that. Only the DHCP client, and the TCP and UDP modules are enabled in this example:

- TCP TX and RX Socket Size = 512
- UDP TX and RX Socket Size = 512
- Built by MPLAB XC32 C/C++ Compiler without any optimization
- Personal Computer IP Address: 172.16.175.86
- PIC32 Ethernet Starter Kit IP Address: 172.16.175.54

UDP Test

PIC32 ESK as UDP server (PIC32 receive):

- Command on the PIC32 device: `iperf -s -u -i 1`



- Command on personal computer: `iperf -c 172.16.175.54 -b 10M -i 1 -t 5`

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

d:\GreenTools>iperf -c 172.16.175.54 -b 10M -i 1 -t 5
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 172.16.175.54, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 63.0 KByte (default)
------------------------------------------------------------
[1908] local 172.16.175.86 port 2598 connected with 172.16.175.54 port 5001
[ ID] Interval       Transfer     Bandwidth
[1908]  0.0- 1.0 sec  1.19 MBytes  10.0 Mbits/sec
[1908]  1.0- 2.0 sec  1.19 MBytes  10.0 Mbits/sec
[1908]  2.0- 3.0 sec  1.19 MBytes  10.0 Mbits/sec
[1908]  3.0- 4.0 sec  1.19 MBytes  10.0 Mbits/sec
[1908]  4.0- 5.0 sec  1.19 MBytes  10.0 Mbits/sec
[1908]  0.0- 5.0 sec  5.96 MBytes  9.97 Mbits/sec
[1908] Server Report:
[1908]  0.0- 5.0 sec  5.95 MBytes  9.99 Mbits/sec  0.000 ms    9/ 4252 (0.21%)
[1908] Sent 4252 datagrams

d:\GreenTools>
```

Personal computer as UDP server (PIC32 transmit)

- Command on personal computer: `iperf -s -u -i 1`

```
C:\WINDOWS\system32\cmd.exe - iperf -s -u -i 1
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

d:\GreenTools>iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[1924] local 172.16.175.86 port 5001 connected with 172.16.175.54 port 9363
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[1924]  0.0- 1.0 sec  2.78 MBytes  23.3 Mbits/sec  0.175 ms    3/ 1984 (0.15%)
[1924]  1.0- 2.0 sec  2.78 MBytes  23.3 Mbits/sec  0.175 ms    0/ 1980 (0%)
[1924]  2.0- 3.0 sec  2.78 MBytes  23.3 Mbits/sec  0.175 ms    0/ 1980 (0%)
[1924]  3.0- 4.0 sec  2.78 MBytes  23.3 Mbits/sec  0.177 ms    0/ 1980 (0%)
[1924]  0.0- 5.0 sec  13.8 MBytes  23.3 Mbits/sec  0.256 ms    3/ 9873 (0.03%)
```

- Command on the PIC32 device: `iperf -c 172.16.175.86 -b 50M -i 1 -t 5`

```
COM9 - PuTTY
==============================================================================

>
> iperf -c 172.16.175.86 -b 50M -i 1 -t 5

>     - RemoteNode MAC: 0 26 2d fe 68 c
------------------------------------------------------------
Client connecting to 172.16.175.86, UDP port 5001

    Session started ...
    - Local  172.16.175.54 port 9363 connected with
    - Remote 172.16.175.86 port 5001
    - Target rate = 50000000 bps, period = 0 ms
    - [ 0-  1 sec]   0/ 1953 ( 0%)    23013 Kbps
    - [ 0-  1 sec]   0/ 1976 ( 0%)    23284 Kbps
    - [ 1-  2 sec]   0/ 1976 ( 0%)    23284 Kbps
    - [ 2-  3 sec]   0/ 1976 ( 0%)    23284 Kbps
    - [ 3-  4 sec]   0/ 1976 ( 0%)    23284 Kbps
    - [0.0- 5.0 sec]   0/ 9873 ( 0%)    23214 Kbps
    ----------------------------------------
    - [0.0- 5.1 sec]   0/ 9882 ( 0%)    22620 Kbps
    Session completed ...
    Tx done. Socket closed.
    Iperf completed.
```

PIC32 ESK as TCP server (PIC32 receiving):

- Command on the PIC32 device: `iperf -s -i 1`

```
COM9 - PuTTY
==============================================================================
* Iperf Command Parser
* (c) 2012, 2013, 2014 -- Microchip Technology, Inc.
*
* Type 'help' to get a list of commands.
==============================================================================

>
> iperf -s -i 1

> ------------------------------------------------------------
Server listening on TCP port 5001

    Session started ...
    - Local  172.16.175.54 port 5001 connected with
    - Remote 172.16.175.86 port 2489
    - [ 0-  1 sec]   0/ 167 ( 0%)     711 Kbps
    - [ 1-  2 sec]   0/ 167 ( 0%)     715 Kbps
    - [ 2-  3 sec]   0/ 167 ( 0%)     715 Kbps
    - [ 3-  4 sec]   0/ 167 ( 0%)     715 Kbps
    - [ 4-  5 sec]   0/ 167 ( 0%)     715 Kbps
    - [ 5-  6 sec]   0/ 144 ( 0%)     617 Kbps
    - [0.0- 6.0 sec]   0/ 979 ( 0%)     691 Kbps
    Session completed ...
```

- Command on personal computer: `iperf -c 172.16.175.54 -i 1 -t 5`

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

d:\GreenTools>iperf -c 172.16.175.54 -i 1 -t 5
------------------------------------------------------------
Client connecting to 172.16.175.54, TCP port 5001
TCP window size: 63.0 KByte (default)
------------------------------------------------------------
[1908] local 172.16.175.86 port 2489 connected with 172.16.175.54 port 5001
[ ID] Interval       Transfer     Bandwidth
[1908]  0.0- 1.0 sec   152 KBytes  1.25 Mbits/sec
[1908]  1.0- 2.0 sec  88.0 KBytes   721 Kbits/sec
[1908]  2.0- 3.0 sec  88.0 KBytes   721 Kbits/sec
[1908]  3.0- 4.0 sec  88.0 KBytes   721 Kbits/sec
[1908]  4.0- 5.0 sec  88.0 KBytes   721 Kbits/sec
[1908]  0.0- 6.1 sec   512 KBytes   690 Kbits/sec

d:\GreenTools>_
```

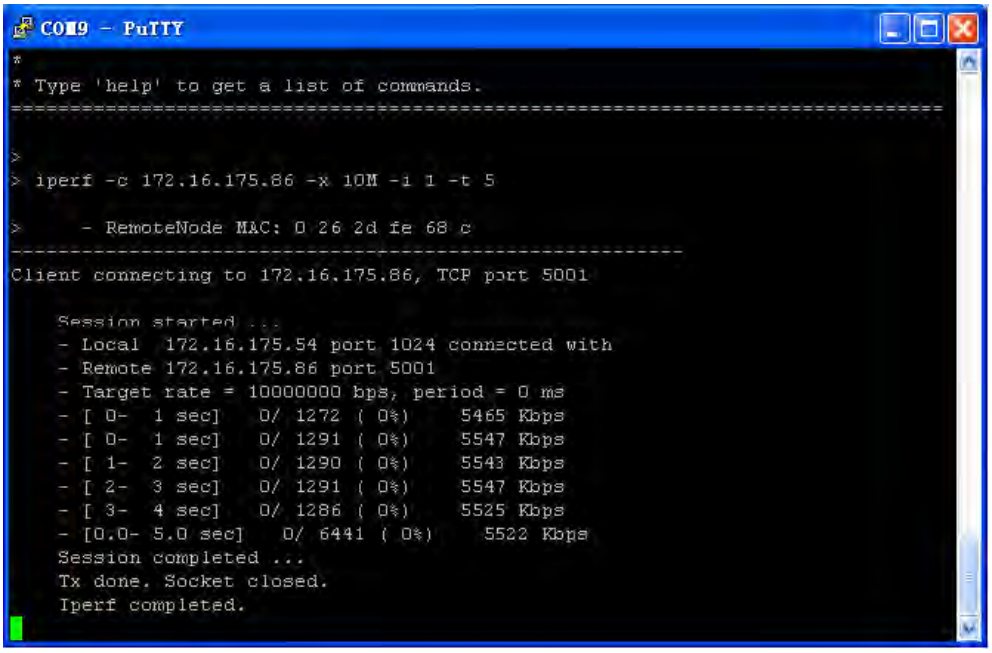
Personal computer as TCP server (PIC32 transmit):

- Command on personal computer: `iperf -s -i 1`

```
C:\WINDOWS\system32\cmd.exe - iperf -s -i 1
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

d:\GreenTools>iperf -s -i 1
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 8.00 KByte (default)
------------------------------------------------------------
[1852] local 172.16.175.86 port 5001 connected with 172.16.175.54 port 9363
[ ID] Interval       Transfer     Bandwidth
[1852]  0.0- 1.0 sec   662 KBytes  5.42 Mbits/sec
[1852]  1.0- 2.0 sec   678 KBytes  5.55 Mbits/sec
[1852]  2.0- 3.0 sec   676 KBytes  5.54 Mbits/sec
[1852]  3.0- 4.0 sec   677 KBytes  5.54 Mbits/sec
[1852]  4.0- 5.0 sec   673 KBytes  5.51 Mbits/sec
[1852]  0.0- 5.0 sec  3.29 MBytes  5.50 Mbits/sec
_
```

- Command on the PIC32 device: `iperf -c 172.16.175.86 -x 10M -i 1 -t 5`



Configuring the Module

Macros

	Name	Description
	TCPIP_IPERF_RX_BUFFER_SIZE	Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_BUFFER_SIZE	Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_QUEUE_LIMIT	for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections
	TCPIP_IPERF_TX_WAIT_TMO	timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth
	TCPIP_IPERF_MAX_INSTANCES	maximum number of simultaneously running iperf instances
	TCPIP_IPERF_TIMING_ERROR_MARGIN	iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0

Description

The configuration of the Iperf TCP/IP Stack is based on the template file `iperf_config.h` with the actual symbols generated in the `system_config.h` file.

This header file contains the configuration selection for the Iperf TCP/IP Stack. Based on the selections made, the Iperf TCP/IP Stack may support the selected features.

TCPIP_IPERF_RX_BUFFER_SIZE Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_RX_BUFFER_SIZE 4096
```

Description

Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers

TCPIP_IPERF_TX_BUFFER_SIZE Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_BUFFER_SIZE 4096
```

Description

Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers

TCPIP_IPERF_TX_QUEUE_LIMIT Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_QUEUE_LIMIT 2
```

Description

for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections

TCPIP_IPERF_TX_WAIT_TMO Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_WAIT_TMO 100
```

Description

timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth

TCPIP_IPERF_MAX_INSTANCES Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_MAX_INSTANCES 1
```

Description

maximum number of simultaneously running iperf instances

TCPIP_IPERF_TIMING_ERROR_MARGIN Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TIMING_ERROR_MARGIN 0
```

Description

iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0

Building the Library

This section lists the files that are available in the Iperf module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.
`/iperf.h`	Header file for the Iperf module.

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/iperf.c`	Iperf 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 Iperf module depends on the following modules:

- TCP/IP Stack Library

Files

Files

Name	Description
iperf_config.h	Iperf configuration file

Description

This section lists the source and header files used by the module.

iperf_config.h

Iperf configuration file

Macros

	Name	Description
	TCPIP_IPERF_MAX_INSTANCES	maximum number of simultaneously running iperf instances
	TCPIP_IPERF_RX_BUFFER_SIZE	Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TIMING_ERROR_MARGIN	iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0

	TCPIP_IPERF_TX_BUFFER_SIZE	Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_QUEUE_LIMIT	for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections
	TCPIP_IPERF_TX_WAIT_TMO	timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth

Description

Iperf Configuration file

This file contains the Iperf configuration options

File Name

iperf_config.h

Company

Microchip Technology Inc.

IPv4 Module

This section describes the TCP/IP Stack Library IPv4 module.

Introduction

TCP/IP Stack Library IPv4 Module for Microchip Microcontrollers

This library provides the API of the IPv4 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

IP is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, and ICMP data gets transmitted as IP datagrams. IP provides an unreliable, connectionless datagram delivery service.

IP provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IP has a simple error handling algorithm: throw away the datagram and try to send an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IP does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IP datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, and then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

Using the Library

This topic describes the basic architecture of the IP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ipv4.h`

The interface to the IPv4 TCP/IP Stack library is defined in the ipv4.h header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the IPv4 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

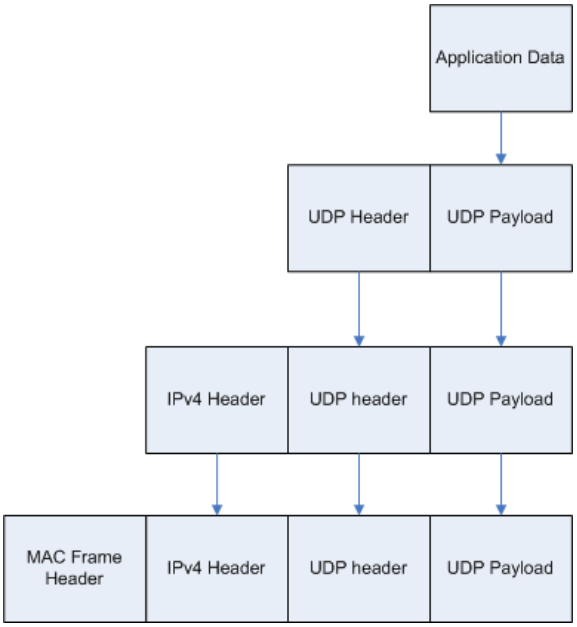
This library provides the API of the IPv4 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the IPv4 module existent in any TCP/IP Stack implementation.

The typical usage of the IP layer in a TCP/IP stack for transmitting a packet is shown in the following diagram:

IPv4 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 IPv4 module.

Library Interface Section	Description
Functions	Routines for configuring IPv4
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the IP TCP/IP Stack Library works.

Description

On the transmit side, the IP layer is responsible for routing and forwarding the IP datagrams to the required destination host. To do that, the IP layer has internal functionality that allows the selection of the proper network interface and destination hardware address of a packet.

Based on the destination of a packet/datagram, the IP layer will send the packet to either a host on the network or to a gateway if either is available. To accomplish this, the IP module works closely with the ARP layer from which it requests the destination hardware addresses for outgoing packets. If the destination address of a packet is not known, the IP module will request an ARP probe and will wait for the ARP module reply. Whenever ARP signals that the solicited address is available, the IP module will send the packet over the network. If a time-out is reported (no such host could be found on that particular network), the packet is silently discarded.

On the receive side, the IP module processes all the incoming IPv4 packets and performs a basic sanity check of the packet checksum and other attributes. If the packet check fails, the packet is simply discarded. Otherwise, it is dispatched for further processing to the appropriate module (UDP, TCP, ICMP, etc.).

The IP layer operation within the TCP/IP stack is transparent to the applications using the preferred socket API (MCHP UDP, TCP or BSD). This means that normally an application does not need to interface directly to the IP layer. However, the IP module exposes an API that can be used by the application for sending IP datagrams directly, bypassing the normal higher layer protocols (UDP, TCP).

Notes:

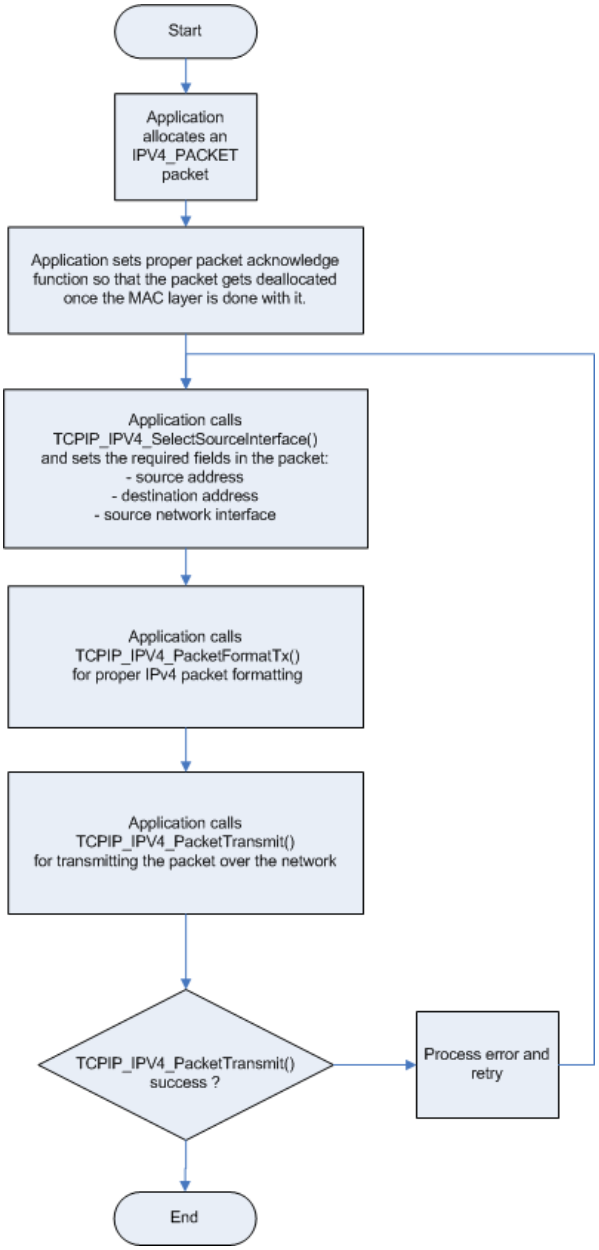
1. Currently, it is not possible for an application to intercept the incoming IP datagrams and interfere with the default IP processing. However, this may change in the future and support can be added for the receive side.
2. For a detailed description of the IP layer functionality, please consult RFC 791, RFC 1122, RFC 2474, etc.
3. This implementation does not support packet fragmentation (neither on transmit or receive).

Core Functionality

This topic describes the core functionality of the IP TCP/IP Stack Library.

Description

IP Connection Flow



To transmit an IP packet over the network, the application has to prepare an IPV4_PACKET data structure and properly format it so that the recipients of this packet (the IP and the MAC layers) can process the packet.

The TCP/IP Stack modules allocate the IPV4_PACKET data structures dynamically using the TCP/IP Stack private heap using packet allocation functions.

Note: Currently, the packet and heap allocation functions are not exposed as public APIs as they are meant for stack's private usage. This may change in the future.

The IPV4_PACKET packet can be allocated statically by the application.

When the IPV4_PACKET is constructed an important thing to remember is to fill in the appropriate packet acknowledge function. Once the MAC layer/driver is done with processing the packet (successfully or not) it will call the IPV4_PACKET acknowledge function, indicating that the packet is now available (and could be freed or reused).

The next step is to fill in the packet source and destination addresses, as well as the outgoing interface (please keep in mind that the TCP/IP stack supports multiple network interfaces).

This could be done with a call to TCPIP_IPV4_SelectSourceInterface or the application can select all these addresses manually.

All that's left before handing the packet is to call TCPIP_IPV4_PacketFormatTx. This call fills in the supplied packet with the IP header structure and updates the IP checksum. Please note that this function expects a properly formatted packet, with the source and destination addresses filled in.

After this, the application can call TCPIP_IPV4_PacketTransmit. This function does some basic packet checks and expects that the packet has a valid network interface selected otherwise the call will fail.

After further formatting (with the MAC required headers), the IP layer will try to send the packet over the network. If the destination hardware

address is known (the ARP resolve call succeeds) the packet is handed over to the MAC. Otherwise, the IP will insert the packet into a queue, waiting for the ARP resolution. When ARP signals that the request is done (either success or timeout) the IP layer intercepts the signal and removes the packet from the wait queue: either hands it over to the MAC for transmission or it discards it if the ARP resolution failed.

The MAC layer/driver in its turn can transmit it immediately, queue the packet for transmission or discard it (if, for example the network link is down). Regardless, once the packet is processed, the packet acknowledge function is called with an updated acknowledge result. This informs the owner of the packet about the result of the operation and also that the packet is no longer in traffic.

Note: Transmission of chained packets is not supported. Each packet has to be transmitted individually.

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV4_FRAGMENT_MAX_NUMBER	Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded
	TCPIP_IPV4_FRAGMENT_MAX_STREAMS	Upper limit for the number of fragmented streams that could be handled simultaneously
	TCPIP_IPV4_FRAGMENT_TIMEOUT	Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds
	TCPIP_IPV4_FRAGMENTATION	If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported
	TCPIP_IPV4_TASK_TICK_RATE	The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Description

The configuration of the IPv4 TCP/IP Stack is based on the file `system_config.h` (which may include `ip_config.h`).

This header file contains the configuration selection for the IPv4 TCP/IP Stack. Based on the selections made, the IPv4 TCP/IP Stack may support the selected features. Currently, there are no configuration parameters for the IP module.

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.

TCPIP_IPV4_FRAGMENT_MAX_NUMBER Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_MAX_NUMBER 4
```

Description

Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded

TCPIP_IPV4_FRAGMENT_MAX_STREAMS Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_MAX_STREAMS 3
```

Description

Upper limit for the number of fragmented streams that could be handled simultaneously

TCPIP_IPV4_FRAGMENT_TIMEOUT Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_TIMEOUT 15
```

Description

Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds

TCPIP_IPV4_FRAGMENTATION Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENTATION 1
```

Description

If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported

TCPIP_IPV4_TASK_TICK_RATE Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_TASK_TICK_RATE 37
```

Description

The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Building the Library

This section lists the files that are available in the IPv4 module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/ipv4.c`	IPv4 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 IPv4 module depends on the following modules:

- TCP/IP Stack Library

Library Interface

a) Functions

	Name	Description
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPV4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address
	TCPIP_IPV4_Task	Standard TCP/IP stack module task function.
	TCPIP_IPV4_PacketFilterClear	Clears the IPV4 packet filters
	TCPIP_IPV4_PacketFilterSet	Sets the IPV4 packet filters
	TCPIP_IPV4_MacPacketTransmit	Transmits a fully constructed MAC packet over the network.
	TCPIP_IPV4_PacketFormatTx	Formats an IPV4 packet and makes it ready for transmission.

b) Data Types and Constants

	Name	Description
	IPV4_HEADER	This is type IPV4_HEADER.
	IPV4_HEADER_TYPE	List of supported protocols.
	IPV4_PACKET	IPv4 packet structure.
	TCPIP_IPV4_FILTER_TYPE	List of supported IPv4 packet filters.
	TCPIP_IPV4_MODULE_CONFIG	This is type TCPIP_IPV4_MODULE_CONFIG.
	IPV4_FRAGMENT_INFO	Structure of an IPv4 header.

Description

This section describes the Application Programming Interface (API) functions of the IPv4 module.

Refer to each section for a detailed description.

a) Functions

TCPIP_IPV4_PacketGetDestAddress Function

Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET

File

ipv4.h

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetDestAddress(TCPIP_MAC_PACKET* pPkt);
```

Returns

- A valid pointer to an IPV4_ADDR - if it succeeds
- 0 - if call failed

Description

The function will return a pointer to where the IPv4 destination address is located in the TCPIP_MAC_PACKET. The TCPIP_MAC_PACKET is

supposed to be a valid IPv4 packet that has destination address properly set.

Remarks

This function is primarily meant for RX packets.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set.

Parameters

Parameters	Description
pPkt	packet to query

Function

const IPV4_ADDR* TCPIP_IPV4_PacketGetDestAddress(TCPIP_MAC_PACKET* pPkt);

TCPIP_IPV4_PacketGetSourceAddress Function

Returns the IPv4 source address associated with a TCPIP_MAC_PACKET

File

ipv4.h

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetSourceAddress(TCPIP_MAC_PACKET* pPkt);
```

Returns

- A valid pointer to an IPV4_ADDR - if it succeeds
- 0 - if call failed

Description

The function will return a pointer to where the IPv4 source address is located in the TCPIP_MAC_PACKET. The TCPIP_MAC_PACKET is supposed to be a valid IPv4 packet that has properly source address set.

Remarks

This function is primarily meant for RX packets.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set

Parameters

Parameters	Description
pPkt	packet to query

Function

const IPV4_ADDR* TCPIP_IPV4_PacketGetSourceAddress(TCPIP_MAC_PACKET* pPkt);

TCPIP_IPV4_PacketTransmit Function

Transmits an IPV4 packet over the network.

File

ipv4.h

C

```
bool TCPIP_IPV4_PacketTransmit(IPV4_PACKET* pPkt);
```

Returns

- true - if the packet was handed to the MAC or is queued for transmission
- false - the packet cannot be transmitted (wrong interface, etc.)

Description

The IPv4 packet is sent to the MAC for transmission.

Remarks

Only single packets can be transmitted. Chained packets are not supported for now.

Preconditions

pPkt should have been properly formatted with TCPIP_IPV4_PacketFormatTx(). The packet interface should be updated.

Parameters

Parameters	Description
pPkt	the packet to be transmitted

Function

bool TCPIP_IPV4_PacketTransmit(IPV4_PACKET* pPkt);

TCPIP_IPV4_SelectSourceInterface Function

Selects a source address and an interface based on the IPv4 destination address

File

ipv4.h

C

```
TCPIP_NET_HANDLE TCPIP_IPV4_SelectSourceInterface(TCPIP_NET_HANDLE netH, IPV4_ADDR* pDestAddress,
IPV4_ADDR* pSrcAddress, bool srcSet);
```

Returns

- A valid interface - if it succeeds and a valid source interface selected
- 0 - interface selection failed

Description

Updates the pSrcAddress and returns the needed interface, if successful:

- if srcSet == 1 and netH != 0, the function will not change anything
- if srcSet == 1 and netH == 0, the call will never fail it will use whatever value in pSrcAddress (even 0) and will try to come up with an appropriate interface
- if srcSet == 0 and netH == 0, it will use the destination address
- if srcSet == 0 and netH != 0, it will use the address of that interface

Remarks

None.

Preconditions

netH has to be valid (if non-0).

Parameters

Parameters	Description
netH	network interface handle
pDestAddress	pointer to destination address
pSrcAddress	pointer to source address
srcSet	boolean; true if address pointed by pSrcAddress is valid

Function

TCPIP_NET_HANDLE TCPIP_IPV4_SelectSourceInterface(TCPIP_NET_HANDLE netH,
IPV4_ADDR* pDestAddress, IPV4_ADDR* pSrcAddress, bool srcSet)

TCPIP_IPV4_Task Function

Standard TCP/IP stack module task function.

File

ipv4.h

C

```
void TCPIP_IPV4_Task();
```

Returns

None.

Description

This function performs IPv4 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The IPv4 module should have been initialized.

Function

void TCPIP_IPV4_Task(void)

TCPIP_IPV4_PacketFilterClear Function

Clears the IPV4 packet filters

File

ipv4.h

C

```
TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterClear(TCPIP_IPV4_FILTER_TYPE filtType);
```

Returns

- the current value of the IPV4 packet filters after this mask was applied.

Description

The function will clear the IPv4 packet filters. The filters that are present in the mask will be cleared. Other filters won't be touched.

Remarks

None.

Preconditions

filtType - valid IPv4 filter IPv4 properly initialiazed

Parameters

Parameters	Description
filtType	packet filter mask to clear

Function

TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterClear(TCPIP_IPV4_FILTER_TYPE filtType);

TCPIP_IPV4_PacketFilterSet Function

Sets the IPV4 packet filters

File

ipv4.h

C

```
TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterSet(TCPIP_IPV4_FILTER_TYPE filtType);
```

Returns

- the current value of the IPV4 packet filters after this mask was applied.

Description

The function will set the IPv4 packet filters. The filters that are present in the mask will be set. Other filters won't be touched.

Remarks

None.

Preconditions

filtType - valid IPv4 filter IPv4 properly initialiazed

Parameters

Parameters	Description
filtType	packet filter mask to set

Function

TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterSet(TCPIP_IPV4_FILTER_TYPE filtType);

TCPIP_IPV4_MacPacketTransmit Function

Transmits a fully constructed MAC packet over the network.

File

ipv4.h

C

```
bool TCPIP_IPV4_MacPacketTransmit(TCPIP_MAC_PACKET* pPkt, TCPIP_NET_HANDLE hNet, IPV4_ADDR* pDestAddress);
```

Returns

- true - if the packet was handed to the MAC or queued internally
- false - the packet cannot be transmitted (wrong interface, etc.)

Description

The function is a helper to transmit a fully constructed MAC packet over an interface. The MAC packet is supposed to have all the fields properly constructed (including the TCPIP_MAC_ETHERNET_HEADER fields). The MAC packet does not need ARP resolution since the DestMACAddr is already known. The packet acknowledge function should be properly set (it will be called by the MAC driver once the packet is transmitted).

Remarks

Only single packets can be transmitted. Chained packets are not supported for now.

The packet could be composed of multiple fragments. If the transmission fails, some fragments could have been transmitted while other failed. The flag TCPIP_MAC_PKT_FLAG_QUEUED could be used to determine which fragments didn't make it through.

Preconditions

pPkt should have been properly formatted and constructed into a valid MAC packet that can go out over an interface pDestAddress - valid pointer

Parameters

Parameters	Description
pPkt	the packet to be transmitted
hNet	interface over which the packet is to be transmitted
pDestAddress	the IPv4 packet destination address It is used only for routing purposes, to decide if the packet is routed internally or sent to the network interface

Function

bool TCPIP_IPV4_MacPacketTransmit(TCPIP_MAC_PACKET* pPkt, TCPIP_NET_HANDLE hNet, IPV4_ADDR* pDestAddress);

TCPIP_IPV4_PacketFormatTx Function

Formats an IPV4 packet and makes it ready for transmission.

File

ipv4.h

C

```
void TCPIP_IPV4_PacketFormatTx(IPV4_PACKET* pPkt, uint8_t protocol, uint16_t ipLoadLen, uint8_t ttl);
```

Returns

None

Description

The necessary fields are set into the IPv4 packet.

Remarks

The segments should be properly updated with the right number of bytes (segLen). The IP payload length (ipLoadLen) is added only to the 1st segment of the packet! Other segments (for packets having multiple packets) are not touched.

Preconditions

Properly allocated pPkt. The source and destination addresses should be updated in the packet.

Parameters

Parameters	Description
pPkt	the packet to be formatted
protocol	the protocol associated with the packet
ipLoadLen	the IPv4 packet payload length
ttl	the Time to Live of the packet If 0, then the default value will be used Otherwise the passed in parameter

Function

TCPIP_IPV4_PacketFormatTx(IPV4_PACKET* pPkt, uint8_t protocol, uint16_t ipLoadLen, uint8_t ttl);

b) Data Types and Constants

IPV4_HEADER Structure

File

ipv4.h

C

```
typedef struct {
  uint8_t VersionIHL;
  uint8_t TypeOfService;
  uint16_t TotalLength;
  uint16_t Identification;
  IPV4_FRAGMENT_INFO FragmentInfo;
  uint8_t TimeToLive;
  uint8_t Protocol;
  uint16_t HeaderChecksum;
  IPV4_ADDR SourceAddress;
  IPV4_ADDR DestAddress;
} IPV4_HEADER;
```

Description

This is type IPV4_HEADER.

IPV4_HEADER_TYPE Enumeration

List of supported protocols.

File

ipv4.h

C

```
typedef enum {
  IP_PROT_ICMP = (1u),
```

```
  IP_PROT_IGMP = (2u),
  IP_PROT_TCP = (6u),
  IP_PROT_UDP = (17u)
} IPV4_HEADER_TYPE;
```

Description

IPv4 supported protocols

This is the list of the protocols that are supported by this IPv4 implementation.

Remarks

None.

IPV4_PACKET Structure

IPv4 packet structure.

File

ipv4.h

C

```
typedef struct {
  TCPIP_MAC_PACKET macPkt;
  IPV4_ADDR srcAddress;
  IPV4_ADDR destAddress;
  TCPIP_NET_HANDLE netIfH;
  IPV4_ADDR arpTarget;
} IPV4_PACKET;
```

Members

Members	Description
TCPIP_MAC_PACKET macPkt;	standard MAC packet header safe cast to TCPIP_MAC_PACKET
IPV4_ADDR srcAddress;	packet source
IPV4_ADDR destAddress;	packet destination
TCPIP_NET_HANDLE netIfH;	packet interface
IPV4_ADDR arpTarget;	ARP resolution target

Description

IPv4 packet structure definition

This is the structure of an IPv4 packet for transmission over the network.

Remarks

None.

TCPIP_IPV4_FILTER_TYPE Enumeration

List of supported IPv4 packet filters.

File

ipv4.h

C

```
typedef enum {
  TCPIP_IPV4_FILTER_NONE = 0x00,
  TCPIP_IPV4_FILTER_UNICAST = 0x01,
  TCPIP_IPV4_FILTER_BROADCAST = 0x02,
  TCPIP_IPV4_FILTER_MULTICAST = 0x04
} TCPIP_IPV4_FILTER_TYPE;
```

Members

Members	Description
TCPIP_IPV4_FILTER_NONE = 0x00	no packet filter active. All packets are accepted
TCPIP_IPV4_FILTER_UNICAST = 0x01	unicast packets will be filtered out
TCPIP_IPV4_FILTER_BROADCAST = 0x02	broadcast packets will be filtered out

TCPIP_IPV4_FILTER_MULTICAST = 0x04	multicast packets will be filtered out

Description

IPv4 packet filters

This is the list of the packet filters that are supported by this IPv4 implementation. There are 3 types of IPv4 packets currently supported:

- unicast
- broadcast
- multicast

An IPV4 packet is accepted if the filter corresponding to the packet type is not set

Remarks

Multiple filters can be set

If no filter is set, all packets are accepted this is the default case.

TCPIP_IPV4_MODULE_CONFIG Structure

File

ipv4.h

C

```
typedef struct {
} TCPIP_IPV4_MODULE_CONFIG;
```

Description

This is type TCPIP_IPV4_MODULE_CONFIG.

IPV4_FRAGMENT_INFO Union

Structure of an IPv4 header.

File

ipv4.h

C

```
typedef union {
  uint16_t val;
  struct {
    uint16_t fragOffset : 13;
    uint16_t MF : 1;
    uint16_t DF : 1;
    uint16_t reserved : 1;
  }
} IPV4_FRAGMENT_INFO;
```

Members

Members	Description
uint16_t fragOffset : 13;	fragment offset
uint16_t MF : 1;	more fragments
uint16_t DF : 1;	don't fragment

Description

IPv4 packet header definition

This is the structure of an IPv4 packet header.

Remarks

None.

Files

Files

Name	Description
ipv4.h	IPv4 definitions for the Microchip TCP/IP Stack.
ip_config.h	Internet Protocol (IP) Configuration file.

Description

This section lists the source and header files used by the library.

ipv4.h

IPv4 definitions for the Microchip TCP/IP Stack.

Enumerations

	Name	Description
	IPV4_HEADER_TYPE	List of supported protocols.
	TCPIP_IPV4_FILTER_TYPE	List of supported IPv4 packet filters.

Functions

	Name	Description
	TCPIP_IPV4_MacPacketTransmit	Transmits a fully constructed MAC packet over the network.
	TCPIP_IPV4_PacketFilterClear	Clears the IPV4 packet filters
	TCPIP_IPV4_PacketFilterSet	Sets the IPV4 packet filters
	TCPIP_IPV4_PacketFormatTx	Formats an IPV4 packet and makes it ready for transmission.
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPV4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address
	TCPIP_IPV4_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	IPV4_HEADER	This is type IPV4_HEADER.
	IPV4_PACKET	IPv4 packet structure.
	TCPIP_IPV4_MODULE_CONFIG	This is type TCPIP_IPV4_MODULE_CONFIG.

Unions

	Name	Description
	IPV4_FRAGMENT_INFO	Structure of an IPv4 header.

Description

IPv4 Header File

IP is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, and ICMP data gets transmitted as IP datagrams. IP provides an unreliable, connectionless datagram delivery service.

File Name

ipv4.h

Company

Microchip Technology Inc.

ip_config.h

Internet Protocol (IP) Configuration file.

Macros

	Name	Description
	TCPIP_IPV4_FRAGMENT_MAX_NUMBER	Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded
	TCPIP_IPV4_FRAGMENT_MAX_STREAMS	Upper limit for the number of fragmented streams that could be handled simultaneously
	TCPIP_IPV4_FRAGMENT_TIMEOUT	Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds
	TCPIP_IPV4_FRAGMENTATION	If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported
	TCPIP_IPV4_TASK_TICK_RATE	The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Description

This file contains the IP module configuration options.

File Name

ip_config.h

Company

Microchip Technology Inc.

IPv6 Module

This section describes the TCP/IP Stack Library IPv6 module.

Introduction

TCP/IP Stack Library IPv6 Module for Microchip Microcontrollers

This library provides the API of the IPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

IPv6 is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, ICMP, and IGMP data gets transmitted as IP datagrams. IP provides an reliable, connectionless datagram delivery service.

IPv6 provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IPv6 has a simple error handling algorithm: throw away the datagram and try to send an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IPv6 does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IPv6 datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, and then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

Using the Library

This topic describes the basic architecture of the IPv6 TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ipv6.h`

The interface to the IPv6 TCP/IP Stack library is defined in the `ipv6.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the IPv6 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

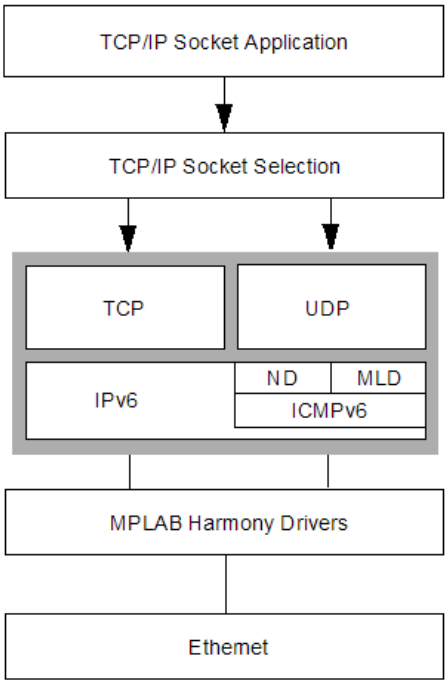
Abstraction Model

This library provides the API of the IP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IP Software Abstraction Block Diagram

This module provides software abstraction of the IP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6 module.

Library Interface Section	Description
Functions	Routines for configuring IPv6
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE	Sets the minimum allocation unit for the payload size
	TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME	Default 30 seconds , Router advertisement reachable time
	TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT	IPv4 Time-to-Live parameter
	TCPIP_IPV6_DEFAULT_LINK_MTU	Default Maximum Transmission Unit
	TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME	1 second, Process the router advertisement's retransmission time
	TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT	Fragmentation packet time-out value. default value is 60 .
	TCPIP_IPV6_INIT_TASK_PROCESS_RATE	IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.
	TCPIP_IPV6_MINIMUM_LINK_MTU	Sets the lower bounds of the Maximum Transmission Unit
	TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT	10 minutes
	TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT	This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out

	TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE	RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8 bytes) + ICMPv6 Echo request header(8 bytes) • Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed then this... more
	TCPIP_IPV6_TASK_PROCESS_RATE	IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_ULA_NTP_ACCESS_TMO	NTP access time-out for the IPv6 ULA address generation, ms
	TCPIP_IPV6_ULA_NTP_VALID_WINDOW	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Description

The configuration of the IPv6 TCP/IP Stack is based on the file `system_config.h`.

This header file contains the configuration selection for the IPv6 TCP/IP Stack. Based on the selections made, the IPv6 TCP/IP Stack may support the support selected features. These configuration settings will apply to all instances of the IPv6 TCP/IP Stack.

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.

TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE (64u)
```

Description

Sets the minimum allocation unit for the payload size

TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME 30u
```

Description

Default 30 seconds , Router advertisement reachable time

TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT 64u
```

Description

IPv4 Time-to-Live parameter

TCPIP_IPV6_DEFAULT_LINK_MTU Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_LINK_MTU 1500u
```

Description

Default Maximum Transmission Unit

TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME 1000u
```

Description

1 second, Process the router advertisement's retransmission time

TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT 60
```

Description

Fragmentation packet time-out value. default value is 60 .

TCPIP_IPV6_INIT_TASK_PROCESS_RATE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_INIT_TASK_PROCESS_RATE (32)
```

Description

IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.

TCPIP_IPV6_MINIMUM_LINK_MTU Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_MINIMUM_LINK_MTU 1280u
```

Description

Sets the lower bounds of the Maximum Transmission Unit

TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT 600ul          // 10 minutes
```

Description

10 minutes

TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT 4
```

Description

This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.

TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT 1
```

Description

This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.

TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT 10u
```

Description

This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out

TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE 1514
```

Description

RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8

bytes) + ICMPv6 Echo request header(8 bytes)

- Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed

then this following macro should be rectified to get proper ICMPv6 ECHO response. This is the Maximum RX fragmented buffer size.

TCPIP_IPV6_TASK_PROCESS_RATE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_TASK_PROCESS_RATE (1000)
```

Description

IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_IPV6_ULA_NTP_ACCESS_TMO Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_ULA_NTP_ACCESS_TMO (12000)
```

Description

NTP access time-out for the IPv6 ULA address generation, ms

TCPIP_IPV6_ULA_NTP_VALID_WINDOW Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_ULA_NTP_VALID_WINDOW (1000)
```

Description

the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Building the Library

This section lists the files that are available in the IPv6 module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/ipv6.c	IPv6 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 IPv6 module depends on the following modules:

- TCP/IP Stack Library
- NDP Module
- SNTP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for a IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for a IPv6 packet.
	TCPIP_IPV6_Flush	Flushes a IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfaceIsReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.
	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for a IPv6 packet.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_RouterAddressAdd	Adds a new router address to a specified interface.
	TCPIP_IPV6_DefaultRouterDelete	Deletes the current router list for a specified interface
	TCPIP_IPV6_DefaultRouterGet	Returns the current router address for a specified interface.
	TCPIP_IPV6_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	IP_VERSION_4	Using IPv4
	IP_VERSION_6	Using IPv6
	IPV6_DATA_DYNAMIC_BUFFER	Data to transmit is allocated in dynamically allocated RAM
	IPV6_DATA_NETWORK_FIFO	Data to transmit is stored in the Network Controller's FIFOs
	IPV6_DATA_NONE	The data segment is unused

	IPV6_DATA_PIC_RAM	Data to transmit is stored in PIC RAM
	IPV6_HEADER_OFFSET_DEST_ADDR	Header offset for destination address
	IPV6_HEADER_OFFSET_NEXT_HEADER	Header offset for next header
	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH	Header offset for payload length
	IPV6_HEADER_OFFSET_SOURCE_ADDR	Header offset for source address
	IPV6_NO_UPPER_LAYER_CHECKSUM	Value flag for no upper layer checksum
	IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM	IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option
	IPV6_TLV_HBHO_ROUTER_ALERT	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	IPV6_TLV_PAD_1	IPv6 Type-length-value type code for the Pad 1 option
	IPV6_TLV_PAD_N	IPv6 Type-length-value type code for the Pad N option
	IPV6_TLV_UNREC_OPT_DISCARD_PP	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address
	IPV6_TLV_UNREC_OPT_DISCARD_SILENT	IPv6 action code for the unrecognized option reaction to discard the packet silently
	IPV6_TLV_UNREC_OPT_SKIP_OPTION	IPv6 action code for the unrecognized option reaction to skip the option
	TCPIP_IPV6_PutArray	Writes data to a packet
	IPV6_ACTION	Provides a list of possible IPv6 actions.
	IPV6_ADDRESS_POLICY	Data structure for IPv6 address policy.
	IPV6_ADDRESS_PREFERENCE	Provides selection of public versus temporary addresses.
	IPV6_ADDRESS_TYPE	Data structure for IPv6 address types.
	IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	_IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	IPV6_EVENT_HANDLER	Clients can register a handler with the IPv6 service.
	IPV6_EVENT_TYPE	This enumeration is used to notify IPv6 client applications.
	IPV6_FRAGMENT_HEADER	Data structure for IPv6 fragment header.
	IPV6_HANDLE	Pointer to IPv6 object
	IPV6_HEADER	IPv6 packet header definition.
	IPV6_NEXT_HEADER_TYPE	Defines a list of IPv6 next header types.
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	_IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	IPV6_PACKET_ACK_FNC	Packet allocation and deallocation acknowledgment callback function.
	IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	_IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.
	IPV6_ULA_FLAGS	Provides a list of possible ULA action flags.
	IPV6_ULA_RESULT	Provides a list of possible ULA results.
	TCPIP_IPV6_MODULE_CONFIG	Provides a place holder for IPv6 configuration.

Description

This section describes the Application Programming Interface (API) functions of the IPv6 module.

Refer to each section for a detailed description.

a) Functions

TCPIP_IPV6_AddressUnicastRemove Function

Removed a configured unicast address from an interface.

File

ipv6.h

C

```
void TCPIP_IPV6_AddressUnicastRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address);
```

Returns

None.

Description

This function is used to remove a configured unicast address from an interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface to remove the address
address	The address

Function

void TCPIP_IPV6_AddressUnicastRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

TCPIP_IPV6_ArrayGet Function

Reads the next byte of data from the specified MAC.

File

ipv6.h

C

```
uint8_t TCPIP_IPV6_ArrayGet(TCPIP_MAC_PACKET* pRxPkt, uint8_t * val, uint16_t len);
```

Returns

- > 0 - The number of bytes read
- 0 - No byte is available to read

Description

Reads a character of data from a packet.

Remarks

None

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from
val	The buffer to store the data
len	The amount of data to read

Function

uint8_t TCPIP_IPV6_ArrayGet (TCPIP_MAC_PACKET* pRxPkt, uint8_t *val,
uint16_t len);

TCPIP_IPV6_ArrayPutHelper Function

Helper function to write data to a packet.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_ArrayPutHelper(IPV6_PACKET * pkt, const void * dataSource, uint8_t dataType,
unsigned short len);
```

Returns

unsigned short - The number of bytes of data written.

Description

This is a helper function for writing data to a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet
dataSource	The address of the data on its medium
dataType	Descriptor of the data type (dynamic memory on PIC, in a network FIFO, in static PIC RAM)
len	Length of the data

Function

unsigned short TCPIP_IPV6_ArrayPutHelper (IPV6_PACKET * ptrPacket,

const void * dataSource, uint8_t dataType, unsigned short len)

TCPIP_IPV6_DASSourceAddressSelect Function

Determines the appropriate source address for a given destination address.

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_DASSourceAddressSelect(TCPIP_NET_HANDLE hNetIf, const IPV6_ADDR * dest,
IPV6_ADDR * requestedSource);
```

Returns

- On Success - Pointer to the selected source address
- On Failure - NULL

Description

The IPv6 policy table is used to select the destination address. The destination address selection algorithm takes a list of IPv6 addresses (gPolicyTable) and sorts the linked list. There are eight sorting rules. Starting with the last rule and working to the most important, using a stable sorting algorithm, will produce a sorted list most efficiently. The best average run time we'll get with a stable sort with O(1) memory usage is O(n^2), so we'll use an insertion sort. This will usually be most efficient for small lists (which should be the typical case). If a rule determines a result, then the remaining rules are not relevant and should be ignored.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
hNetIf	The given interface

dest	The destination address
requestedSource	A specified source

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_DASSourceAddressSelect (TCPIP_NET_HANDLE hNetIf, IPV6_ADDR * dest, IPV6_ADDR * requestedSource)

TCPIP_IPV6_DestAddressGet Function

Gets the destination address for a IPv6 packet.

File

ipv6.h

C

```
IPV6_ADDR * TCPIP_IPV6_DestAddressGet(IPV6_PACKET * p);
```

Returns

IPV6_ADDR *

- On Success - Get a valid IPv6 Destination address
- On Failure - NULL

Description

This function is used to get the destination address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Function

IPV6_ADDR * TCPIP_IPV6_DestAddressGet(IPV6_PACKET * p)

TCPIP_IPV6_DestAddressSet Function

Sets the destination address for a IPv6 packet.

File

ipv6.h

C

```
void TCPIP_IPV6_DestAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr);
```

Returns

None.

Description

This API is used to configure the destination address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addr	destination address

Function

void TCPIP_IPV6_DestAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr)

TCPIP_IPV6_Flush Function

Flushes a IP TX packet.

File

ipv6.h

C

```
int TCPIP_IPV6_Flush(IPV6_PACKET * pkt);
```

Returns

- 1 - if the packet has been transmitted
- 0 - if the packet has been queued
- <0 - if the packet has been discarded for some error

Description

This function flushes a IP TX packet. Determines the link-layer address, if necessary and calculates the upper-layer checksum, if necessary.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to flush

Function

int TCPIP_IPV6_Flush (IPV6_PACKET * ptrPacket)

TCPIP_IPV6_Get Function

Reads the next byte of data from the specified MAC.

File

ipv6.h

C

```
uint8_t TCPIP_IPV6_Get(TCPIP_MAC_PACKET* pRxPkt, uint8_t* pData);
```

Returns

- 1 - On Successful read
- 0 - No byte is available to read

Description

This function is used to read the next byte of data from a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from
pData	Byte data to be read

Function

uint8_t TCPIP_IPV6_Get (TCPIP_MAC_PACKET* pRxPkt, uint8_t* pData)

TCPIP_IPV6_HandlerDeregister Function

Deregisters an IPv6 event handler callback function.

File

ipv6.h

C

```
bool TCPIP_IPV6_HandlerDeregister(IPV6_HANDLE hIpv6);
```

Returns

- true - if deregister successful
- false - if deregister is not successful

Description

This function deregisters a previously registered IPv6 event handler callback function.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
hIpv6	Handle to registered callback

Function

bool TCPIP_IPV6_HandlerDeregister(IPV6_HANDLE hIpv6)

TCPIP_IPV6_HandlerRegister Function

Registers an IPv6 event handler callback function.

File

ipv6.h

C

```
IPV6_HANDLE TCPIP_IPV6_HandlerRegister(TCPIP_NET_HANDLE hNet, IPV6_EVENT_HANDLER handler, const void*
hParam);
```

Returns

Handle to registered callback.

- On Success - Returns a valid handle
- On Failure - null handle

Description

This function is used to register a notification handler with the IPv6 module.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	Specifies interface to register on.
handler	Handler to be called for event.
hParam	The hParam is passed by the client and will be used by the IPv6 when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Function

IPV6_HANDLE TCPIP_IPV6_HandlerRegister(TCPIP_NET_HANDLE hNet,
IPV6_EVENT_HANDLER handler, const void* hParam)

TCPIP_IPV6_InterfaceIsReady Function

Determines if an interface is ready for IPv6 transactions.

File

ipv6.h

C

```
bool TCPIP_IPV6_InterfaceIsReady(TCPIP_NET_HANDLE netH);
```

Returns

- true - if the interface has IPv6 functionality available
- false - if the interface does not have IPv6 functionality available

Description

Returns the current state of the IPv6 interface (i.e., determines if an interface is ready for IPv6 transactions).

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
netH	The interface to check

Function

bool TCPIP_IPV6_InterfaceIsReady (TCPIP_NET_HANDLE netH)

TCPIP_IPV6_MulticastListenerAdd Function

Adds a multicast listener to an interface.

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_MulticastListenerAdd(TCPIP_NET_HANDLE hNet, IPV6_ADDR * address);
```

Returns

IPV6_ADDR_STRUCT *

- On Success - Pointer to the new listener
- On Failure - NULL

Description

This function is used to add the IPv6 multicast address to an interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
hNet	The interface to add the address
address	The new listener

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_MulticastListenerAdd(TCPIP_NET_HANDLE hNet, IPV6_ADDR * address)

TCPIP_IPV6_MulticastListenerRemove Function

Removes a multicast listener from a given interface.

File

ipv6.h

C

```
void TCPIP_IPV6_MulticastListenerRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address);
```

Returns

None.

Description

This function removes a multicast listener from a given interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface
address	The address

Function

void TCPIP_IPV6_MulticastListenerRemove (TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

TCPIP_IPV6_PacketFree Function

Frees a TCP/IP Packet structure from dynamic memory.

File

ipv6.h

C

```
void TCPIP_IPV6_PacketFree(IPV6_PACKET * pkt);
```

Returns

None.

Description

This function frees a TCP/IP Packet structure from dynamic memory.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to free.

Function

void TCPIP_IPV6_PacketFree (IPV6_PACKET * ptrPacket)

TCPIP_IPV6_PayloadSet Function

Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_PayloadSet(IPV6_PACKET * pkt, uint8_t* payload, unsigned short len);
```

Returns

- On Success - The amount of data added to the packet length
- On Failure - 0

Description

This function will allocate a data segment header and append it to the end of a chain of segments in a TX packet. It will set the data pointer in the packet segment to a pre-existing buffer of data.

Remarks

This function is useful for adding payloads to outgoing packets without copying them if the data is in another pre-existing buffer (i.e., TCP).

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet
payload	Address of the data payload
len	Length of the data payload

Function

unsigned short TCPIP_IPV6_PayloadSet (IPV6_PACKET * ptrPacket, uint8_t* payload,

unsigned short len)

TCPIP_IPV6_Put Function

Writes a character of data to a packet.

File

ipv6.h

C

```
bool TCPIP_IPV6_Put(IPV6_PACKET * pkt, unsigned char v);
```

Returns

- true - if the character was written
- false - if the character was not written

Description

This function writes a character of data to a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pkt	The packet
v	The character

Function

bool TCPIP_IPV6_Put (IPV6_PACKET * pkt, unsigned char v)

TCPIP_IPV6_SourceAddressGet Function

Gets the source address for an IPv6 packet.

File

ipv6.h

C

```
IPV6_ADDR * TCPIP_IPV6_SourceAddressGet(IPV6_PACKET * p);
```

Returns

IPV6_ADDR *

- On Success - Get a valid IPv6 Source address
- On Failure - NULL

Description

This API is used to get the source address for an IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Function

IPV6_ADDR * TCPIP_IPV6_SourceAddressGet(IPV6_PACKET * p)

TCPIP_IPV6_SourceAddressSet Function

Sets the source address for a IPv6 packet.

File

ipv6.h

C

```
void TCPIP_IPV6_SourceAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr);
```

Returns

None.

Description

This function is used to configure the source address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addr	source address

Function

void TCPIP_IPV6_SourceAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr)

TCPIP_IPV6_TxIsPutReady Function

Determines whether a TX packet can be written to.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_TxIsPutReady(IPV6_PACKET * pkt, unsigned short count);
```

Returns

- On Success - The amount of space available
- On Failure - 0

Description

This function determines whether a TX packet can be written to. This function will allocate additional space to the packet to accommodate the user.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to check
count	The amount of writable space to check for

Function

unsigned short TCPIP_IPV6_TxIsPutReady (IPV6_PACKET * ptrPacket,
unsigned short count)

TCPIP_IPV6_TxPacketAllocate Function

Dynamically allocates a packet for transmitting IP protocol data.

File

ipv6.h

C

```
IPV6_PACKET * TCPIP_IPV6_TxPacketAllocate(TCPIP_NET_HANDLE netH, IPV6_PACKET_ACK_FNC ackFnc, void*
ackParam);
```

Returns

- On Success - Pointer to the allocated packet
- On Failure - NULL

Description

This function dynamically allocates a packet for transmitting IP protocol data and sets the packet IPv6 protocol for a TX packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	Interface of the outgoing packet.
ackFnc	function to be called when IP is done with the TX packet (finished transmitting)
ackParam	parameter to be used for this callback This has meaning only for the caller of the TCPIP_IPV6_TxPacketAllocate

Function

IPV6_PACKET * TCPIP_IPV6_TxPacketAllocate (TCPIP_NET_HANDLE netH,
IPV6_PACKET_ACK_FNC ackFnc, void* ackParam);

TCPIP_IPV6_UniqueLocalUnicastAddressAdd Function

Adds a Unique Local Unicast Address (ULA) to a specified interface.

File

ipv6.h

C

```
IPV6_ULA_RESULT TCPIP_IPV6_UniqueLocalUnicastAddressAdd(TCPIP_NET_HANDLE netH, uint16_t subnetID,
IPV6_ULA_FLAGS genFlags, IP_MULTI_ADDRESS* ntpAddress);
```

Returns

- IPV6_ULA_RES_BUSY - address generation module is busy
- IPV6_ULA_RES_IF_ERR - IPv6 interface is not up
- IPV6_ULA_RES_OK - if the call succeeded and the generation process was started

Description

This function starts the process of adding an ULA address to the specified interface. The segments of the generated address are as follows:

- FC00::/7 - ULA prefix
- L - 1 bit set to 1, locally assigned
- Global ID - 40 bit random generated identifier
- subnet ID - 16 bit subnet identifier
- Interface ID - 64 bit interface identifier generated as a EUI64 from the specified interface MAC

The randomness of the "Global ID" prefix of the generated IPv6 address is obtained by using an NTP server. The supplied NTP server will be

contacted to obtain an NTP time stamp. This time stamp together with the EUI64 identifier obtained from the interface MAC are passed through a 160 bits hash algorithm (SHA1) and the least significant 40 bits are used as the GlobalID of the interface.

Remarks

Only one address generation at a time is supported for now. Before attempting a new address generation the previous operation has to be completed, otherwise the call will fail.

This function requires that the NTP client is enabled in the stack. If not, the call will fail.

The caller will be notified by the outcome of the operation by the stack using the standard IPv6 notification handler (registered by TCPIP_IPV6_HandlerRegister call).

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface to add the address to.
subnetID	The subnet ID to be used.
genFlags	Address generation flags: • IPV6_ULA_FLAG_NTPV4 - if set, the ntpAddress is an IPv4 address and the NTP server will be contacted over an IPv4 connection. Otherwise, a default IPv6 connection will be attempted • IPV6_ULA_FLAG_GENERATE_ONLY - if set, the address will not be added to the list of the addresses for the specified interface • IPV6_ULA_FLAG_SKIP_DAD - if set, the DAD processing will be skipped
ntpAddress	The NTP server address - it is an IPv4/IPv6 address as selected by the IPV6_ULA_FLAG_NTPV4 flag (the IP address could be obtained with an DNS call into the stack)

Function

IPV6_ULA_RESULT TCPIP_IPV6_UniqueLocalUnicastAddressAdd (TCPIP_NET_HANDLE netH, uint16_t subnetID, IPV6_ULA_FLAGS genFlags, IP_MULTI_ADDRESS* ntpAddress)

TCPIP_IPV6_UnicastAddressAdd Function

Adds a unicast address to a specified interface

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_UnicastAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * address, int prefixLen,
uint8_t skipProcessing);
```

Returns

IPV6_ADDR_STRUCT *

- On Success - Pointer to the structure of the newly allocated address
- On Failure - NULL

Description

Adds a unicast address to a specified interface. Starts duplicate address detection if necessary.

Remarks

The RFC (4291) requires the interface ID for all unicast addresses, (except those that start with the binary value 000) to be 64 bits long and to be constructed in Modified EUI-64 format. Therefore the prefixLen parameter should probably always be 64.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface to which the the address is to be addded

address	The address to add
prefixLen	The prefix length associated to this static address (providing the subnet prefix length). If 0, the default value of 64 will be used
skipProcessing	true to skip Duplicate address detection; otherwise, false

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_UnicastAddressAdd (TCPIP_NET_HANDLE netH,
IPV6_ADDR * address, int prefixLen, uint8_t skipProcessing)

TCPIP_IPV6_RouterAddressAdd Function

Adds a new router address to a specified interface.

File

ipv6.h

C

```
bool TCPIP_IPV6_RouterAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * rAddress, unsigned long validTime, int flags);
```

Returns

- true - operation succeeded
- false - operation failed (interface not valid, interface still configuring, no memory, etc.)

Description

This function adds a unicast address as a routing address to a specified interface.

Remarks

The validTime parameter is relevant for an existent router on the network. If such router does not exist the stack will eventually discard the entry automatically.

Preconditions

rAddress pointer to a valid IPv6 router address IPv6 stack initialized, interface up and configured

Parameters

Parameters	Description
netH	The interface to which the address is to be added
rAddress	The router address to add
validTime	The time this valid will be valid, seconds. If 0, the entry will be valid forever.
flags	Creation flags (not used for now, should be 0)

Function

bool TCPIP_IPV6_RouterAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * rAddress,
unsigned long validTime, int flags);

TCPIP_IPV6_DefaultRouterDelete Function

Deletes the current router list for a specified interface

File

ipv6.h

C

```
void TCPIP_IPV6_DefaultRouterDelete(TCPIP_NET_HANDLE netH);
```

Returns

None.

Description

This function deletes all default routers on a given interface

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface for which to delete the router address.

Function

void TCPIP_IPV6_DefaultRouterDelete(TCPIP_NET_HANDLE netH);

TCPIP_IPV6_DefaultRouterGet Function

Returns the current router address for a specified interface.

File

ipv6.h

C

```
const IPV6_ADDR* TCPIP_IPV6_DefaultRouterGet(TCPIP_NET_HANDLE netH);
```

Returns

- valid IPV6_ADDR pointer - if the interface exists and a valid router address exists
- 0 - if the interface does not exist and a valid router address does not exist

Description

This function returns the current router address for a specified interface.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface for which to return the router address.

Function

const IPV6_ADDR* TCPIP_IPV6_DefaultRouterGet(TCPIP_NET_HANDLE hNet);

TCPIP_IPV6_Task Function

Standard TCP/IP stack module task function.

File

ipv6.h

C

```
void TCPIP_IPV6_Task();
```

Returns

None.

Description

This function performs IPv6 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The IPv6 stack is initialized.

Function

void TCPIP_IPV6_Task(void)

b) Data Types and Constants

IP_VERSION_4 Macro

File

ipv6.h

C

```
#define IP_VERSION_4 (0u) // Using IPv4
```

Description

Using IPv4

IP_VERSION_6 Macro

File

ipv6.h

C

```
#define IP_VERSION_6 (1u) // Using IPv6
```

Description

Using IPv6

IPV6_DATA_DYNAMIC_BUFFER Macro

File

ipv6.h

C

```
#define IPV6_DATA_DYNAMIC_BUFFER (0x1u)        // Data to transmit is allocated in dynamically allocated RAM
```

Description

Data to transmit is allocated in dynamically allocated RAM

IPV6_DATA_NETWORK_FIFO Macro

File

ipv6.h

C

```
#define IPV6_DATA_NETWORK_FIFO (0x2u)        // Data to transmit is stored in the Network Controller's FIFOs
```

Description

Data to transmit is stored in the Network Controller's FIFOs

IPV6_DATA_NONE Macro

File

ipv6.h

C

```
#define IPV6_DATA_NONE (0x0u)        // The data segment is unused
```

Description

The data segment is unused

IPV6_DATA_PIC_RAM Macro

File

ipv6.h

C

```
#define IPV6_DATA_PIC_RAM (0x3u)        // Data to transmit is stored in PIC RAM
```

Description

Data to transmit is stored in PIC RAM

IPV6_HEADER_OFFSET_DEST_ADDR Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_DEST_ADDR (0x08u + sizeof (IPV6_ADDR))
```

Description

Header offset for destination address

IPV6_HEADER_OFFSET_NEXT_HEADER Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_NEXT_HEADER (0x06u)
```

Description

Header offset for next header

IPV6_HEADER_OFFSET_PAYLOAD_LENGTH Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_PAYLOAD_LENGTH (0x04u)
```

Description

Header offset for payload length

IPV6_HEADER_OFFSET_SOURCE_ADDR Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_SOURCE_ADDR (0x08u)
```

Description

Header offset for source address

IPV6_NO_UPPER_LAYER_CHECKSUM Macro

File

ipv6.h

C

```
#define IPV6_NO_UPPER_LAYER_CHECKSUM (0xFFFFu)    // Value flag for no upper layer checksum
```

Description

Value flag for no upper layer checksum

IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM Macro

File

ipv6.h

C

```
#define IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM 0xC2u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option

IPV6_TLV_HBHO_ROUTER_ALERT Macro

File

ipv6.h

C

```
#define IPV6_TLV_HBHO_ROUTER_ALERT 0x05u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option

IPV6_TLV_PAD_1 Macro

File

ipv6.h

C

```
#define IPV6_TLV_PAD_1 0u
```

Description

IPv6 Type-length-value type code for the Pad 1 option

IPV6_TLV_PAD_N Macro

File

ipv6.h

C

```
#define IPV6_TLV_PAD_N 1u
```

Description

IPv6 Type-length-value type code for the Pad N option

IPV6_TLV_UNREC_OPT_DISCARD_PP Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP 0b10
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message

IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC 0b11
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address

IPV6_TLV_UNREC_OPT_DISCARD_SILENT Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_SILENT 0b01
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet silently

IPV6_TLV_UNREC_OPT_SKIP_OPTION Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_SKIP_OPTION 0b00
```

Description

IPv6 action code for the unrecognized option reaction to skip the option

TCPIP_IPV6_PutArray Macro

Writes data to a packet

File

ipv6.h

C

```
#define TCPIP_IPV6_PutArray(pkt,data,len) TCPIP_IPV6_ArrayPutHelper(pkt, data, IPV6_DATA_PIC_RAM, len)
```

Returns

unsigned short - The number of bytes of data written.

Description

This function writes data to an outgoing packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true. The TCPIP_IPV6_TxIsPutReady function must have returned a value greater than or equal to 'len'.

Parameters

Parameters	Description
ptrPacket	The packet
dataSource	Pointer to the data to copy to the packet
len	Length of the data

Function

unsigned short TCPIP_IPV6_PutArray (IPV6_PACKET * ptrPacket,
const void * dataSource, unsigned short len)

IPV6_ACTION Enumeration

Provides a list of possible IPv6 actions.

File

ipv6.h

C

```
typedef enum {
  IPV6_ACTION_NONE = 0,
  IPV6_ACTION_DISCARD_SILENT,
  IPV6_ACTION_DISCARD_PP_0,
  IPV6_ACTION_DISCARD_PP_2,
  IPV6_ACTION_DISCARD_PP_2_NOT_MC,
  IPV6_ACTION_BEGIN_EX_HEADER_PROCESSING
} IPV6_ACTION;
```

Members

Members	Description
IPV6_ACTION_NONE = 0	No action. Skip the option
IPV6_ACTION_DISCARD_SILENT	Discard the packet silently
IPV6_ACTION_DISCARD_PP_0	Discard the packet and send an ICMP parameter problem message with code value 0
IPV6_ACTION_DISCARD_PP_2	Discard the packet and send an ICMP parameter problem message with code value 2
IPV6_ACTION_DISCARD_PP_2_NOT_MC	Discard the packet and send an ICMP parameter problem message with code value 3
IPV6_ACTION_BEGIN_EX_HEADER_PROCESSING	Begin extension processing

Description

Enumeration: IPV6_ACTION

Different actions need to be taken depending on the result of our header processing.

Remarks

None.

IPV6_ADDRESS_POLICY Structure

Data structure for IPv6 address policy.

File

ipv6.h

C

```
typedef struct {
  IPV6_ADDR address;
  unsigned char prefixLength;
  unsigned char precedence;
  unsigned char label;
} IPV6_ADDRESS_POLICY;
```

Members

Members	Description
IPV6_ADDR address;	IPv6 address
unsigned char prefixLength;	IPv6 prefix length
unsigned char precedence;	IPv6 address precedence
unsigned char label;	IPv6 label type

Description

Type: IPV6_ADDRESS_POLICY

The policy table is the longest matching prefix lookup table and is used to select the destination IPv6 Address.

If Precedence(A) > Precedence(B), address A has higher precedence than address B. The label value Label(A) allows for policies that prefer a particular source address prefix for use with a destination address prefix. Default Policy table - Prefix Precedence Label ::1/128 50 0 ::/0 40 1 2002::/16 30 2 ::/96 20 3 ::ffff:0.0.0.0/96 10 4

Remarks

None.

Example

const IPV6_ADDRESS_POLICY gPolicyTable[] = { {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01}}, 128, 50, 0}, // Loopback address {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 0, 40, 1}, // Unspecified address {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00}}, 96, 35, 4}, // IPv4-mapped address {{{0x20, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 16, 30, 2}, // 2002::/15 - 6to4 {{{0x20, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 32, 5, 5}, // 2001::/32 - Teredo tunneling {{{0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 7, 3, 13}, // ULA };

IPV6_ADDRESS_PREFERENCE Enumeration

Provides selection of public versus temporary addresses.

File

ipv6.h

C

```
typedef enum {
  IPV6_PREFER_PUBLIC_ADDRESSES = 0,
  IPV6_PREFER_TEMPORARY_ADDRESSES
} IPV6_ADDRESS_PREFERENCE;
```

Description

Enumeration: IPV6_ADDRESS_PREFERENCE

None

Remarks

None.

IPV6_ADDRESS_TYPE Union

Data structure for IPv6 address types.

File

ipv6.h

C

```
typedef union {
  unsigned char byte;
  struct {
    unsigned scope : 4;
    unsigned type : 2;
  } bits;
} IPV6_ADDRESS_TYPE;
```

Description

Type: IPV6_ADDRESS_TYPE

This type defines the data structure for IPv6 address types.

Remarks

None.

IPV6_DATA_SEGMENT_HEADER Structure

Data structure for IPv6 Data Segment header.

File

ipv6.h

C

```
typedef struct _IPV6_DATA_SEGMENT_HEADER {
  uint8_t* dataLocation;
  unsigned short segmentSize;
  unsigned short segmentLen;
  unsigned char memory;
  unsigned char segmentType;
  struct _IPV6_DATA_SEGMENT_HEADER * nextSegment;
  void * data[0];
} IPV6_DATA_SEGMENT_HEADER;
```

Members

Members	Description
uint8_t* dataLocation;	Location of the data to transmit
unsigned short segmentSize;	Size of this data segment
unsigned short segmentLen;	Number of bytes of data in this segment
unsigned char memory;	Type: IPV6_DATA_NONE, IPV6_DATA_DYNAMIC_BUFFER, IPV6_DATA_NETWORK_FIFO, IPV6_DATA_PIC_RAM
unsigned char segmentType;	Type of segment contents
struct _IPV6_DATA_SEGMENT_HEADER * nextSegment;	Pointer to the next data segment
void * data[0];	Optional buffer space

Description

Structure : IPV6_DATA_SEGMENT_HEADER

Data structure is used to allocate a data segment header and an optional payload.

Remarks

None.

IPV6_EVENT_HANDLER Type

Clients can register a handler with the IPv6 service.

File

ipv6.h

C

```
typedef void (* IPV6_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, IPV6_EVENT_TYPE evType, const void* evParam,
const void* usrParam);
```

Description

Type: IPV6_EVENT_HANDLER

Once an IPv6 event occurs the IPv6 service will call the registered handler The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

evParam is a parameter that's associated to an IPv6 event

- For an address event (IPV6_EVENT_ADDRESS_ADDED, IPV6_EVENT_ADDRESS_REMOVED) it should typecast to (const IPV6_ADDR_STRUCT*)
- For an IPV6_EVENT_ULA_ADDRESS_GENERATED ULA event it should typecast to (const IPV6_ADDR*)
- For an IPV6_EVENT_ULA_ADDRESS_FAILED ULA event the evParam is an IPV6_ULA_RESULT error code

The evParam is invalid outside of the IPV6_EVENT_HANDLER context call and should not be stored by the caller. Info that's needed has to be copied into caller's own context.

usrParam is a user-supplied parameter

Remarks

For address related events the passed (const IPV6_ADDR_STRUCT*) parameter is invalid after the notification call.

IPV6_EVENT_TYPE Enumeration

This enumeration is used to notify IPv6 client applications.

File

ipv6.h

C

```
typedef enum {
  IPV6_EVENT_ADDRESS_ADDED = 1,
  IPV6_EVENT_ADDRESS_REMOVED,
  IPV6_EVENT_ULA_ADDRESS_GENERATED,
  IPV6_EVENT_ULA_ADDRESS_FAILED
} IPV6_EVENT_TYPE;
```

Members

Members	Description
IPV6_EVENT_ADDRESS_ADDED = 1	Event is generated when IPv6 added to the list
IPV6_EVENT_ADDRESS_REMOVED	Event is generated when IPv6 address removed from the list
IPV6_EVENT_ULA_ADDRESS_GENERATED	Event is generated when IPv6 unique local address included to the list
IPV6_EVENT_ULA_ADDRESS_FAILED	Event is generated when IPv6 unique local address is failed

Description

Enumeration: IPV6_EVENT_TYPE

These events are used while notifying to the registered applications.

Remarks

None.

IPV6_FRAGMENT_HEADER Structure

Data structure for IPv6 fragment header.

File

ipv6.h

C

```
typedef struct {
  uint8_t nextHeader;
  uint8_t reserved;
  union {
    struct {
      uint16_t m : 1;
      uint16_t reserved2 : 2;
      uint16_t fragmentOffset : 13;
    } bits;
    uint16_t w;
  } offsetM;
  uint32_t identification;
} IPV6_FRAGMENT_HEADER;
```

Description

Structure: IPV6_FRAGMENT_HEADER

The Fragment header is used by an IPv6 source to send a packet larger than would fit in the path MTU to its destination. The Fragment header is identified by a Next Header value of 44 in the immediately preceding header.

Remarks

None.

IPV6_HANDLE Type

File

ipv6.h

C

```
typedef const void * IPV6_HANDLE;
```

Description

Pointer to IPv6 object

IPV6_HEADER Structure

IPv6 packet header definition.

File

ipv6.h

C

```
typedef struct {
  unsigned long V_T_F;
  unsigned short PayloadLength;
  unsigned char NextHeader;
  unsigned char HopLimit;
  IPV6_ADDR SourceAddress;
  IPV6_ADDR DestAddress;
} IPV6_HEADER;
```

Members

Members	Description
unsigned long V_T_F;	Version , Traffic class and Flow Label
unsigned short PayloadLength;	Length of IPv6 payload, i.e. the rest of packet following this IPv6 header in octets
unsigned char NextHeader;	Identifies the type of header immediately following IPv6 header
unsigned char HopLimit;	Decremented by 1 by each node that forwards the packet
IPV6_ADDR SourceAddress;	128 bit address of originator
IPV6_ADDR DestAddress;	128 bit address of intended recipient

Description

Structure: IPV6_HEADER

IPv6 packet header definition.

Remarks

Any extension headers present are considered part of the payload length.

IPV6_NEXT_HEADER_TYPE Enumeration

Defines a list of IPv6 next header types.

File

ipv6.h

C

```
typedef enum {
  IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u),
  IPV6_PROT_ICMP = (1u),
  IPV6_PROT_TCP = (6u),
  IPV6_PROT_UDP = (17u),
  IPV6_PROT_IPV6 = (41u),
  IPV6_PROT_ROUTING_HEADER = (43u),
  IPV6_PROT_FRAGMENTATION_HEADER = (44u),
  IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u),
  IPV6_PROT_AUTHENTICATION_HEADER = (51u),
  IPV6_PROT_ICMPV6 = (58u),
  IPV6_PROT_NONE = (59u),
  IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)
} IPV6_NEXT_HEADER_TYPE;
```

Members

Members	Description
IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u)	IPv6 Hop-by-Hop Opt. Header
IPV6_PROT_ICMP = (1u)	ICMPv4 Header
IPV6_PROT_TCP = (6u)	TCP protocol Header
IPV6_PROT_UDP = (17u)	UDP protocol Header
IPV6_PROT_IPV6 = (41u)	IPv6 Protocol
IPV6_PROT_ROUTING_HEADER = (43u)	IPv6 Routing Header
IPV6_PROT_FRAGMENTATION_HEADER = (44u)	IPv6 Fragmentation Header
IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u)	Encapsulating Security Payload Header
IPV6_PROT_AUTHENTICATION_HEADER = (51u)	Authentication Header
IPV6_PROT_ICMPV6 = (58u)	ICMPv6 Protocol
IPV6_PROT_NONE = (59u)	No next header
IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)	Destination Options Header

Description

Enumeration: IPV6_NEXT_HEADER_TYPE

Identifies the type of the next header immediately follows IPv6 header.

Remarks

None

IPV6_PACKET Structure

Packet structure/state tracking for IPv6 packets.

File

ipv6.h

C

```
typedef struct _IPV6_PACKET {
  struct _IPV6_PACKET * next;
  unsigned short payloadLen;
  unsigned short headerLen;
  unsigned short upperLayerHeaderLen;
  unsigned short upperLayerChecksumOffset;
  unsigned char upperLayerHeaderType;
  union {
    struct {
      unsigned char reserved : 3;
      unsigned char useUnspecAddr : 1;
      unsigned char sourceSpecified : 1;
      unsigned char queued : 1;
      unsigned char addressType : 2;
    }
    unsigned char val;
  } flags;
  TCPIP_MAC_ADDR remoteMACAddr;
  IPV6_PACKET_ACK_FNC ackFnc;
  TCPIP_MAC_PACKET_ACK_FUNC macAckFnc;
  void* ackParam;
  void* clientData;
  void * neighbor;
  unsigned short offsetInSegment;
  uint32_t queuedPacketTimeout;
  IPV6_DATA_SEGMENT_HEADER payload;
  TCPIP_NET_HANDLE netIfH;
  IPV6_HEADER ipv6Header;
} IPV6_PACKET;
```

Members

Members	Description
struct _IPV6_PACKET * next;	Next packet in a queue
unsigned short payloadLen;	Amount of data in payload buffer
unsigned short headerLen;	Total header length (IP header + IPv6 Extension headers)
unsigned short upperLayerHeaderLen;	Total length of the upper layer header
unsigned short upperLayerChecksumOffset;	Offset of the upper layer checksum
unsigned char upperLayerHeaderType;	Type definition for the upper-layer header type
unsigned char useUnspecAddr : 1;	This packet should use the unspecified address
unsigned char sourceSpecified : 1;	The upper layer or application layer specified a source address
unsigned char queued : 1;	Packet has been queued
unsigned char addressType : 2;	IP_ADDRESS_TYPE_IPV6 or IP_ADDRESS_TYPE_IPV4
TCPIP_MAC_ADDR remoteMACAddr;	The packet's remote MAC address
IPV6_PACKET_ACK_FNC ackFnc;	function to be called when IPv6 done with the packet
TCPIP_MAC_PACKET_ACK_FUNC macAckFnc;	function to be called when MAC done with a TX packet
void* ackParam;	parameter to be used
void* clientData;	optional packet client data
void * neighbor;	The neighbor that the message was received from
unsigned short offsetInSegment;	Offset used for storing fragment transmission information
uint32_t queuedPacketTimeout;	Time out for IPv6 packets which are queued
IPV6_DATA_SEGMENT_HEADER payload;	IPv6 data segment payload
TCPIP_NET_HANDLE netIfH;	packet network interface
IPV6_HEADER ipv6Header;	IPv6 header definition

Description

Structure: IPV6_PACKET

IPv6 packets are queued for future transmission.queuedPacketTimeout is used to time-out IPv6 queued packets.

Remarks

For IPv6 queuing the time out has to be 0! The queue is processed separately by the NDP.IPV6_PACKET_ACK_FNC is called after the successful removal and inclusion of the packet.

IPV6_PACKET_ACK_FNC Type

Packet allocation and deallocation acknowledgment callback function.

File

ipv6.h

C

```
typedef bool (* IPV6_PACKET_ACK_FNC)(void*, bool, const void*);
```

Returns

-true - if the packet needs the queuing flags removed (it is not deleted and still in use) -false - if the packet does not need the queuing flags removed (either no longer exists or the flags updated)

Description

Type: IPV6_PACKET_ACK_FNC

Packet allocation and deallocation acknowledgment callback function.

Remarks

None.

Parameters

Parameters	Description
void *	Pointer to the packet that was transmitted
bool	True if the packet was sent, false otherwise
const void*	0

IPV6_RX_FRAGMENT_BUFFER Structure

Data structure for IPv6 Received fragmented packet.

File

ipv6.h

C

```
typedef struct _IPV6_RX_FRAGMENT_BUFFER {
  struct _IPV6_RX_FRAGMENT_BUFFER * next;
  uint8_t * ptrPacket;
  uint32_t identification;
  uint16_t bytesInPacket;
  uint16_t packetSize;
  uint16_t firstFragmentLength;
  uint8_t secondsRemaining;
} IPV6_RX_FRAGMENT_BUFFER;
```

Members

Members	Description
struct _IPV6_RX_FRAGMENT_BUFFER * next;	Next fragmented packet
uint8_t * ptrPacket;	Packet information
uint32_t identification;	Fragment id
uint16_t bytesInPacket;	Number of bytes written to packet
uint16_t packetSize;	Packet size (packet is complete when this matches bytesInPacket)
uint16_t firstFragmentLength;	Length of the first fragment
uint8_t secondsRemaining;	Number of seconds remaining during which the fragment can be reassembled

Description

Structure: IPV6_RX_FRAGMENT_BUFFER

Each fragment is composed of unfragmentable Part and fragmentable part. Allocate memory for the fragmented packet w.r.t to TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE. The RFC specifies that the fragments must be reassembled in one minute or less.secondsRemaining is the times remaining for reassemble.

Remarks

None.

IPV6_SEGMENT_TYPE Enumeration

Provides an enumeration of IPv6 segment types.

File

ipv6.h

C

```
typedef enum {
  TYPE_IPV6_HEADER = 1u,
  TYPE_IPV6_EX_HEADER_HOP_BY_HOP_OPTIONS,
  TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_1,
  TYPE_IPV6_EX_HEADER_ROUTING,
  TYPE_IPV6_EX_HEADER_FRAGMENT,
  TYPE_IPV6_EX_HEADER_AUTHENTICATION_HEADER,
  TYPE_IPV6_EX_HEADER_ENCAPSULATING_SECURITY_PAYLOAD,
  TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_2,
  TYPE_IPV6_UPPER_LAYER_HEADER,
  TYPE_IPV6_UPPER_LAYER_PAYLOAD,
  TYPE_IPV6_BEGINNING_OF_WRITABLE_PART,
  TYPE_IPV6_END_OF_LIST
} IPV6_SEGMENT_TYPE;
```

Description

Enumeration: IPV6_SEGMENT_TYPE

IPv6 extended header.

Remarks

None.

IPV6_TLV_OPTION_TYPE Union

Data structure for IPv6 TLV options.

File

ipv6.h

C

```
typedef union {
  unsigned char b;
  struct {
    unsigned option : 6;
    unsigned unrecognizedAction : 2;
  } bits;
} IPV6_TLV_OPTION_TYPE;
```

Description

Type: IPV6_TLV_OPTION_TYPE

This type defines the data structure for IPv6 TLV options.

Remarks

None.

IPV6_ULA_FLAGS Enumeration

Provides a list of possible ULA action flags.

File

ipv6.h

C

```
typedef enum {
  IPV6_ULA_FLAG_NTPV4 = 0x01,
  IPV6_ULA_FLAG_GENERATE_ONLY = 0x02,
  IPV6_ULA_FLAG_SKIP_DAD = 0x04
} IPV6_ULA_FLAGS;
```

Members

Members	Description
IPV6_ULA_FLAG_NTPV4 = 0x01	use an IPv4 NTP server access in Unique Local Address generation default is an IPv6 server
IPV6_ULA_FLAG_GENERATE_ONLY = 0x02	generate an address only, don't add it to the interface addresses
IPV6_ULA_FLAG_SKIP_DAD = 0x04	when adding the address to the interface, skip the Duplicate Address Detection

Description

Enumeration: IPV6_ULA_FLAGS

This enumeration provides a list of possible flags for the Unique Local Address (ULA) generation.

Remarks

None.

IPV6_ULA_RESULT Enumeration

Provides a list of possible ULA results.

File

ipv6.h

C

```
typedef enum {
  IPV6_ULA_RES_OK,
  IPV6_ULA_RES_BUSY = -1,
  IPV6_ULA_RES_IF_ERR = -2,
  IPV6_ULA_RES_NTP_ACCESS_ERR = -3,
  IPV6_ULA_RES_NTP_TSTAMP_ERR = -4
} IPV6_ULA_RESULT;
```

Members

Members	Description
IPV6_ULA_RES_OK	the address generation was started successfully
IPV6_ULA_RES_BUSY = -1	address generation module is busy
IPV6_ULA_RES_IF_ERR = -2	interface non existent
IPV6_ULA_RES_NTP_ACCESS_ERR = -3	NTP module could not be accessed
IPV6_ULA_RES_NTP_TSTAMP_ERR = -4	wrong NTP time stamp received

Description

Enumeration: IPV6_ULA_RESULT

This enumeration provides a list of possible results for the Unique Local Address (ULA) generation.

Remarks

None.

TCPIP_IPV6_MODULE_CONFIG Structure

Provides a place holder for IPv6 configuration.

File

ipv6.h

C

```
typedef struct {
  uint32_t rxfragmentBufSize;
```

```
  uint32_t fragmentPktRxTimeout;
} TCPIP_IPV6_MODULE_CONFIG;
```

Members

Members	Description
uint32_t rxfragmentBufSize;	RX fragmented buffer size
uint32_t fragmentPktRxTimeout;	fragmented packet time out value

Description

Structure: TCPIP_IPV6_MODULE_CONFIG

IPv6 module runtime configuration and initialization parameters.

Remarks

None.

Files

Files

Name	Description
ipv6.h	IPv6 is the next generation TCP/IP protocol suite.
ipv6_config.h	IPv6 configuration file

Description

This section lists the source and header files used by the library.

ipv6.h

IPv6 is the next generation TCP/IP protocol suite.

Enumerations

	Name	Description
	IPV6_ACTION	Provides a list of possible IPv6 actions.
	IPV6_ADDRESS_PREFERENCE	Provides selection of public versus temporary addresses.
	IPV6_EVENT_TYPE	This enumeration is used to notify IPv6 client applications.
	IPV6_NEXT_HEADER_TYPE	Defines a list of IPv6 next header types.
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_ULA_FLAGS	Provides a list of possible ULA action flags.
	IPV6_ULA_RESULT	Provides a list of possible ULA results.

Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DefaultRouterDelete	Deletes the current router list for a specified interface
	TCPIP_IPV6_DefaultRouterGet	Returns the current router address for a specified interface.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for a IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for a IPv6 packet.
	TCPIP_IPV6_Flush	Flushes a IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfaceIsReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.

	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_RouterAddressAdd	Adds a new router address to a specified interface.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for a IPv6 packet.
	TCPIP_IPV6_Task	Standard TCP/IP stack module task function.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface.

Macros

	Name	Description
	IP_VERSION_4	Using IPv4
	IP_VERSION_6	Using IPv6
	IPV6_DATA_DYNAMIC_BUFFER	Data to transmit is allocated in dynamically allocated RAM
	IPV6_DATA_NETWORK_FIFO	Data to transmit is stored in the Network Controller's FIFOs
	IPV6_DATA_NONE	The data segment is unused
	IPV6_DATA_PIC_RAM	Data to transmit is stored in PIC RAM
	IPV6_HEADER_OFFSET_DEST_ADDR	Header offset for destination address
	IPV6_HEADER_OFFSET_NEXT_HEADER	Header offset for next header
	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH	Header offset for payload length
	IPV6_HEADER_OFFSET_SOURCE_ADDR	Header offset for source address
	IPV6_NO_UPPER_LAYER_CHECKSUM	Value flag for no upper layer checksum
	IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM	IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option
	IPV6_TLV_HBHO_ROUTER_ALERT	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	IPV6_TLV_PAD_1	IPv6 Type-length-value type code for the Pad 1 option
	IPV6_TLV_PAD_N	IPv6 Type-length-value type code for the Pad N option
	IPV6_TLV_UNREC_OPT_DISCARD_PP	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address
	IPV6_TLV_UNREC_OPT_DISCARD_SILENT	IPv6 action code for the unrecognized option reaction to discard the packet silently
	IPV6_TLV_UNREC_OPT_SKIP_OPTION	IPv6 action code for the unrecognized option reaction to skip the option
	TCPIP_IPV6_PutArray	Writes data to a packet

Structures

	Name	Description
	_IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	_IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	_IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	IPV6_ADDRESS_POLICY	Data structure for IPv6 address policy.
	IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	IPV6_FRAGMENT_HEADER	Data structure for IPv6 fragment header.
	IPV6_HEADER	IPv6 packet header definition.
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	TCPIP_IPV6_MODULE_CONFIG	Provides a place holder for IPv6 configuration.

Types

	Name	Description
	IPV6_EVENT_HANDLER	Clients can register a handler with the IPv6 service.
	IPV6_HANDLE	Pointer to IPv6 object

	IPV6_PACKET_ACK_FNC	Packet allocation and deallocation acknowledgment callback function.

Unions

	Name	Description
	IPV6_ADDRESS_TYPE	Data structure for IPv6 address types.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.

Description

IPv6 Definitions for the Microchip TCP/IP Stack

Like IPv4 , IPv6 (Internet Protocol Version 6) is the another version of IP protocol. This is the next-generation protocol which works parallel with IPv4. Microchip TCP/IP stack is a dual stack architecture where both Ipv4 and Ipv6 works simultaneously. (RFC - 2460,3484)

File Name

ipv6.h

Company

Microchip Technology Inc.

ipv6_config.h

IPv6 configuration file

Macros

	Name	Description
	TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE	Sets the minimum allocation unit for the payload size
	TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME	Default 30 seconds , Router advertisement reachable time
	TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT	IPv4 Time-to-Live parameter
	TCPIP_IPV6_DEFAULT_LINK_MTU	Default Maximum Transmission Unit
	TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME	1 second, Process the router advertisement's retransmission time
	TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT	Fragmentation packet time-out value. default value is 60 .
	TCPIP_IPV6_INIT_TASK_PROCESS_RATE	IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.
	TCPIP_IPV6_MINIMUM_LINK_MTU	Sets the lower bounds of the Maximum Transmission Unit
	TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT	10 minutes
	TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT	This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out
	TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE	RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8 bytes) + ICMPv6 Echo request header(8 bytes) • Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed then this... more
	TCPIP_IPV6_TASK_PROCESS_RATE	IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_ULA_NTP_ACCESS_TMO	NTP access time-out for the IPv6 ULA address generation, ms

	TCPIP_IPV6_ULA_NTP_VALID_WINDOW	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Description

Internet Control Message Protocol for IPv6 (ICMPv6) Configuration file

This file contains the IPv6 module configuration options

File Name

ipv6_config.h

Company

Microchip Technology Inc.

MAC Driver Module

This section describes the TCP/IP Stack Library MAC Driver module.

Introduction

TCP/IP Stack Library Media Access Control (MAC) Driver Module for Microchip Microcontrollers

This library provides the API of the MAC Driver module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

In the seven-layer OSI model of computer networking, media access control (MAC) data communication protocol is a sub-layer of the data link layer, which itself is layer 2. The MAC sub-layer provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium (e.g., Ethernet). The hardware that implements the MAC is referred to as a medium access controller.

The MAC sub-layer acts as an interface between the logical link control (LLC) sub-layer and the network's physical layer. The MAC layer emulates a full-duplex logical communication channel in a multi-point network. This channel may provide unicast, multicast or broadcast communication service.

Using the Library

This topic describes the basic architecture of the TCP/IP Stack MAC Driver Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_mac.h`

The interface to the TCP/IP Stack MAC Driver Library is defined in the `tcpip_mac.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the TCP/IP Stack MAC Driver Library should include `tcpip.h`.

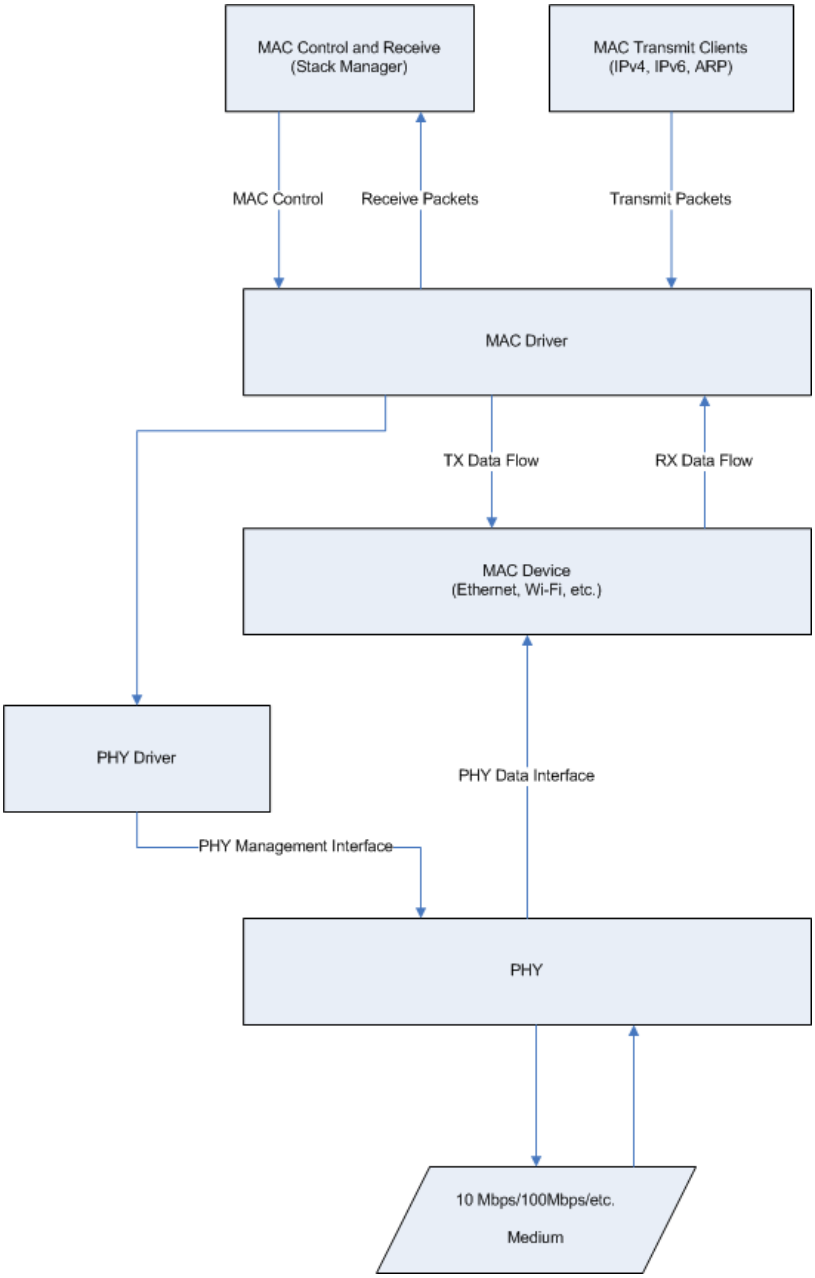
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the TCP/IP Stack MAC module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

MAC 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 MAC module.

Library Interface Section	Description
Initialization Functions	This section provides routines for configuring the MAC driver
Client Control and Status Functions	This section provides routines for client instance manipulations and MAC status
Event Functions	This section provides routines for control of the MAC driver events
Filter Functions	This section provides routines for controlling the receive MAC filtering
Data Transfer Functions	This section provides routines for transmitting and receiving data packets over the network
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the TCP/IP Stack MAC Library works.

Description

The MAC layer (together with the Logical Link Control – LLC) is intended to have the functionality described in the OSI model for the Data Link Layer.

There are two important functions to be performed in the MAC layer as part of the Data Link Layer:

Data Encapsulation (Transmit and Receive)

- Framing (frame boundary detection, frame synchronization)
- Addressing (control of the source and destination addresses)
- Error detection (detection of transmission errors occurring in the physical medium)

Media Access Management

- Medium allocation (collision avoidance)
- Contention resolution (collision handling)

Beside this, the receive packet filtering is another important functionality that is usually integrated in the MAC layer.

The functionality previously presented is handled by the hardware itself as an intrinsic part of the data exchange that's performed with the PHY.

From the software point of view the MAC driver exposes an API that's focused on efficient data transfer routines. Control of the driver and access to the transferred data is given by using regular driver client access functions.

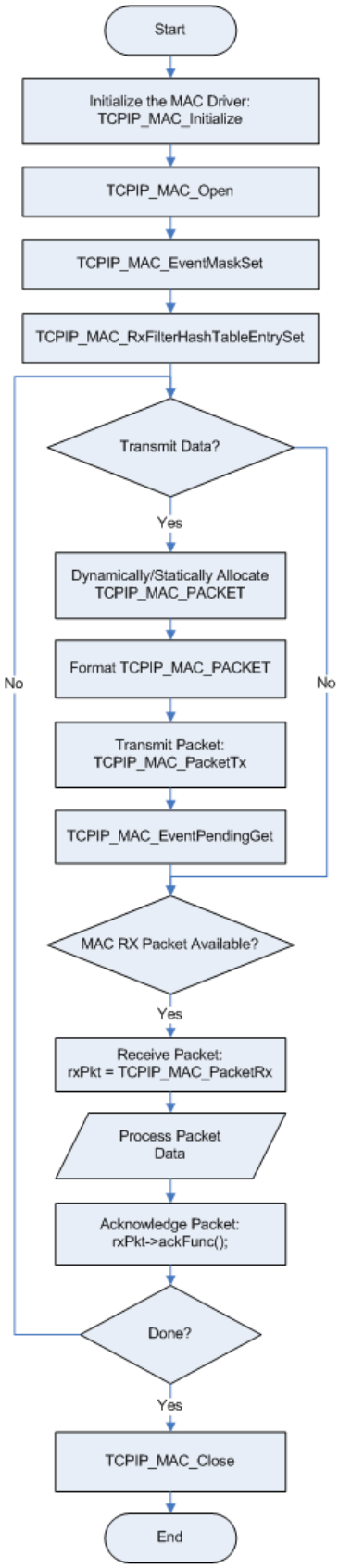
Please note that the MAC model described here corresponds to a virtual MAC driver that the TCP/IP stack uses. All the implementations of the physical MAC drivers (Ethernet, Wi-Fi, etc.) must adhere and implement this API required by the virtual MAC API. This is how the TCP/IP stack achieves virtualization of the operation over multiple network interfaces. The TCP/IP stack makes no assumption about the specifics of an actual implementation and behaves all MACs to behave identically

Core Functionality

This topic describes the core functionality of the TCP/IP Stack MAC Library.

Description

MAC Connection Flow Diagram



The MAC driver is used internally by the TCP/IP Stack Manager and its API is not normally exposed outside the stack. There is little need to interface directly with the MAC from the application level. The TCP/IP Stack provides all the data interface functions to the application by exposing the socket API (TCP and UDP) and also low level IPv4 and IPv6 API. However, the functionality of the MAC driver is important for having a thorough understanding of how the TCP/IP Stack works or for integrating the MPLAB Harmony MAC driver into a third-party TCP/IP Stack.

The MAC driver needs to be initialized by a call to TCPIP_MAC_Initialize() with both stack and module specific initialization data. The stack

initialization data contains absolutely necessary data for the MAC driver to operate (memory allocation functions, packet allocation functions, event notification functions, etc.) Note that the module initialization data could be different from one MAC module to another and, if missing, normally the MAC driver should use some default values specified in the `tcpip_mac_config.h` file.

Once the initialization process succeeds, a MAC client handle is obtained by using TCPIP_MAC_Open.

Other parameters can be set, like the active events and receive filters by calling TCPIP_MAC_EventMaskSet, TCPIP_MAC_RxFilterHashTableEntrySet, etc.

By default the MAC driver operates in interrupts and the events are reported to the TCP/IP stack by using the event notification function that's passed in at the MAC driver initialization. However, by using the TCPIP_MAC_EventPendingGet function, it is possible to use the MAC driver in a polling manner.

To send a packet, a MAC driver client has to obtain a TCPIP_MAC_PACKET data structure, either statically or dynamically. Note that the Harmony TCP/IP Stack uses dynamic packet allocation and has a packet allocation module that the Stack Manager calls for this purpose. The packet has to be properly formatted with all the required fields before being passed to the MAC driver for transmission. The MAC driver will perform a minimum sanity check and will try to transmit the packet over the network. Once the corresponding interrupt signals to the driver that the packet was (successfully/non-successfully) transmitted the MAC driver will call the packet acknowledge function. This will indicate to the owner of the packet (the module that created/allocated the packet) that the packet is no longer in use and it can be reused, discarded, etc.

The receive process is configured as part of the MAC driver initialization procedure. Usually the MAC driver needs some buffers for storing the received network traffic. How exactly this is done is driver specific and not part of this specification. The Harmony Ethernet MAC driver, for example, allocates multiple receive buffers at initialization and uses those for the whole duration of the driver life. Other implementations may choose to free the receive buffers once they are processed and allocate new ones, as needed.

The receive process is initiated by the hardware indicating through an interrupt that a packet is pending. This will trigger a notification event to the MAC client (the TCP/IP stack) and the actual packet receive function can be called. Alternatively the MAC client can call the TCPIP_MAC_EventPendingGet function which can be used in a polling manner.

To actually receive the packet the TCP/IP stack should call the TCPIP_MAC_PacketRx function. This function, besides returning a pointer to the newly received packet will provide additional info about the packet status. Also, some MAC drivers have the possibility to mark the packet being received as being unicast, multicast or broadcast.

Once a packet is obtained, the stack has to dispatch it accordingly, based on the info contained within the packet (ARP, IPv4, IPv6, etc.). Note that in the Harmony TCP/IP stack it's the Stack Manager that performs the receiving and dispatching function of the packets. The recipient of the packet will be called to process the incoming packet and may forward the packet for further processing (TCP, UDP, ICMP). Note that during all this processing time this packet can no longer be used by the MAC driver for storing newly receive data. Therefore once the final recipient of the received packet has processed the data it has to call the packet acknowledge function, so that the packet can be returned to its owner, the AMC driver in this case. If the MAC driver reuses the packet (as the Harmony Ethernet MAC driver does) or simply discards it is driver implementation specific.

Once the data transfer of packets over the network is no longer needed the TCPIP_MAC_Close function can be called and also TCPIP_MAC_Deinitialize. This is part of the normal procedure that the Harmony stack uses for shutting down and restarting a network interface dynamically.

For detailed information about the MAC layer requirements and functionality, please consult the IEEE 802.3 specifications.

Configuring the Library

Macros

	Name	Description
	TCPIP_EMAC_ETH_OPEN_FLAGS	Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value
	TCPIP_EMAC_PHY_ADDRESS	The PHY address, as configured on the board. By default all the PHYs respond to address 0
	TCPIP_EMAC_PHY_CONFIG_FLAGS	Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting
	TCPIP_EMAC_PHY_LINK_INIT_DELAY	The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.
	TCPIP_EMAC_RX_BUFF_SIZE	Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DESCRIPTORS	Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers

	TCPIP_EMAC_RX_FRAGMENTS	MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	TCPIP_EMAC_RX_MAX_FRAME	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).
	TCPIP_EMAC_TX_DESCRIPTORS	Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.
	TCPIP_EMAC_RX_DEDICATED_BUFFERS	Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_INIT_BUFFERS	Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.
	TCPIP_EMAC_RX_LOW_FILL	Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers
	TCPIP_EMAC_RX_LOW_THRESHOLD	Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.
	TCPIP_EMAC_RX_FILTERS	MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC

Description

The configuration of the MAC TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the MAC TCP/IP Stack. Based on the selections made, the MAC TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the MAC TCP/IP Stack.

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.

TCPIP_EMAC_ETH_OPEN_FLAGS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_ETH_OPEN_FLAGS (TCPIP_ETH_OPEN_DEFAULT)
```

Description

Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value

TCPIP_EMAC_PHY_ADDRESS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_ADDRESS (0)
```

Description

The PHY address, as configured on the board. By default all the PHYs respond to address 0

TCPIP_EMAC_PHY_CONFIG_FLAGS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_CONFIG_FLAGS (DRV_ETHPHY_CFG_AUTO)
```

Description

Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting

TCPIP_EMAC_PHY_LINK_INIT_DELAY Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_LINK_INIT_DELAY (500)
```

Description

The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.

TCPIP_EMAC_RX_BUFF_SIZE Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_BUFF_SIZE 1536
```

Description

Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.

TCPIP_EMAC_RX_DESCRIPTORS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_DESCRIPTORS 10
```

Description

Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers

TCPIP_EMAC_RX_FRAGMENTS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_FRAGMENTS ((TCPIP_EMAC_RX_MAX_FRAME + (TCPIP_EMAC_RX_BUFF_SIZE -1 )) /
(TCPIP_EMAC_RX_BUFF_SIZE))
```

Description

MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:

TCPIP_EMAC_RX_MAX_FRAME Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_MAX_FRAME 1536
```

Description

Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).

Remarks

Always multiple of 16.

TCPIP_EMAC_TX_DESCRIPTORS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_TX_DESCRIPTORS 8
```

Description

Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.

TCPIP_EMAC_RX_DEDICATED_BUFFERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_DEDICATED_BUFFERS 4
```

Description

Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.

TCPIP_EMAC_RX_INIT_BUFFERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_INIT_BUFFERS 0
```

Description

Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.

TCPIP_EMAC_RX_LOW_FILL Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_LOW_FILL 2
```

Description

Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers

TCPIP_EMAC_RX_LOW_THRESHOLD Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_LOW_THRESHOLD 1
```

Description

Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.

TCPIP_EMAC_RX_FILTERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_FILTERS (TCPIP_MAC_RX_FILTER_TYPE_DEFAULT)
```

Description

MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC

Building the Library

This section lists the files that are available in the MAC Driver module of the TCP/IP Stack Library.

Description

The MAC Driver module is part of the core functionality of the TCP/IP Stack Library. Please refer to the Building the Library topic in the TCP/IP Stack Library section for build information.

Library Interface

a) Initialization Functions

	Name	Description
	TCPIP_MAC_Open	MAC driver open function.
	TCPIP_MAC_Close	MAC driver close function.
	TCPIP_MAC_Initialize	MAC driver initialization function.
	TCPIP_MAC_Deinitialize	MAC driver deinitialization function.
	TCPIP_MAC_Reinitialize	MAC driver reinitialization function.

b) Client Control and Status Functions

	Name	Description
	TCPIP_MAC_ParametersGet	MAC parameter get function.
	TCPIP_MAC_StatisticsGet	Gets the current MAC statistics.
	TCPIP_MAC_LinkCheck	MAC link checking function.
	TCPIP_MAC_Status	Provides the current status of the MAC driver module.
	TCPIP_MAC_Tasks	Maintains the MAC driver's state machine.
	TCPIP_MAC_Process	MAC periodic processing function.
	TCPIP_MAC_RegisterStatisticsGet	Gets the current MAC hardware statistics registers.
	TCPIP_MAC_ConfigGet	Gets the current MAC driver configuration.

c) Event Functions

	Name	Description
	TCPIP_MAC_EventAcknowledge	This function acknowledges a previously reported MAC event.
	TCPIP_MAC_EventMaskSet	MAC events report enable/disable function.
	TCPIP_MAC_EventPendingGet	Returns the currently pending MAC events.

d) Filter Functions

	Name	Description
	TCPIP_MAC_RxFilterHashTableEntrySet	Sets the current MAC hash table receive filter.

e) Data Transfer Functions

	Name	Description
	TCPIP_MAC_PacketTx	MAC driver transmit function.
	TCPIP_MAC_PacketRx	A packet is returned if such a pending packet exists.

f) Data Types and Constants

	Name	Description
	TCPIP_MAC_ACTION	Network interface action for initialization/deinitialization
	TCPIP_MAC_ADDR	Definition of a MAC address.
	TCPIP_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	TCPIP_MAC_ETHERNET_HEADER	Definition of a MAC frame header.
	TCPIP_MAC_EVENT	Defines the possible MAC event types.
	TCPIP_MAC_EventF	MAC event notification handler.
	TCPIP_MAC_HANDLE	Handle to a MAC driver.
	TCPIP_MAC_HEAP_CallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_FreeF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_HANDLE	A handle used for memory allocation functions.
	TCPIP_MAC_HEAP_MallocF	MAC allocation function prototype.
	TCPIP_MAC_MODULE_CTRL	Data structure that's passed to the MAC at the initialization time.
	TCPIP_MAC_PACKET	Forward reference to a MAC packet.
	TCPIP_MAC_PACKET_ACK_FUNC	Prototype of a MAC packet acknowledge function.
	TCPIP_MAC_PACKET_FLAGS	Flags belonging to MAC packet.
	TCPIP_MAC_PACKET_RX_STAT	Status of a received packet.
	TCPIP_MAC_PKT_ACK_RES	List of MAC return codes for a packet acknowledge function.
	TCPIP_MAC_PKT_AckF	MAC packet acknowledge function prototype.
	TCPIP_MAC_PKT_AllocF	MAC packet allocation function prototype.
	TCPIP_MAC_PKT_FreeF	MAC packet free function prototype.
	TCPIP_MAC_POWER_MODE	This is type TCPIP_MAC_POWER_MODE.
	TCPIP_MAC_PROCESS_FLAGS	List of the MAC processing flags.
	TCPIP_MAC_RES	List of return codes from MAC functions.
	TCPIP_MAC_SEGMENT_FLAGS	Flags belonging to MAC data segment.
	TCPIP_MODULE_MAC_MRF24W_CONFIG	This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.
	TCPIP_MODULE_MAC_PIC32INT_CONFIG	Data that's passed to the MAC at initialization time.
	TCPIP_MAC_RX_STATISTICS	MAC receive statistics data gathered at run time.

	TCPIP_MAC_TX_STATISTICS	MAC transmit statistics data gathered at run time.
	TCPIP_MAC_INIT	Contains all the data necessary to initialize the MAC device.
	TCPIP_MAC_PARAMETERS	Data structure that tells the MAC run time parameters.
	TCPIP_MAC_PKT_AckFDbg	This is type TCPIP_MAC_PKT_AckFDbg.
	TCPIP_MAC_PKT_AllocFDbg	This is type TCPIP_MAC_PKT_AllocFDbg.
	TCPIP_MAC_PKT_FreeFDbg	This is type TCPIP_MAC_PKT_FreeFDbg.
	TCPIP_MAC_HEAP_CallocFDbg	This is type TCPIP_MAC_HEAP_CallocFDbg.
	TCPIP_MAC_HEAP_FreeFDbg	This is type TCPIP_MAC_HEAP_FreeFDbg.
	TCPIP_MAC_HEAP_MallocFDbg	This is type TCPIP_MAC_HEAP_MallocFDbg.
	TCPIP_MAC_STATISTICS_REG_ENTRY	Describes a MAC hardware statistics register.
	TCPIP_MAC_TYPE	List of the MAC types.
	TCPIP_MAC_SYNCH_REQUEST	Defines the possible MAC synchronization request types.
	TCPIP_MAC_SynchReqF	MAC synchronization request function definition.
	TCPIP_MODULE_MAC_ID	IDs of the MAC module supported by the stack.
	TCPIP_MODULE_MAC_MRF24WN_CONFIG	This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.
	TCPIP_MAC_RX_FILTER_TYPE	Defines the possible MAC RX filter types.
	TCPIP_MAC_LINK_MTU	MTU size correspondig to the MAC types.

Description

This section describes the Application Programming Interface (API) functions of the MAC Driver module.

Refer to each section for a detailed description.

a) Initialization Functions

TCPIP_MAC_Open Function

MAC driver open function.

File

tcpip_mac.h

C

```
DRV_HANDLE TCPIP_MAC_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- Valid handle - if the open function succeeded
- DRV_HANDLE_INVALID - if an error occurs

Description

This is the function that opens an instance of the MAC. Once a MAC client handle is obtained all the MAC functions can be accessed using that handle.

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

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
drvIndex	identifier for the driver 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 TCPIP_MAC_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

TCPIP_MAC_Close Function

MAC driver close function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Close(DRV_HANDLE hMac);
```

Returns

None.

Description

This is the function that closes an instance of the MAC. All per client data is released and the handle can no longer be used after this function is called.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	MAC client handle obtained by a call to TCPIP_MAC_Open.

Function

void TCPIP_MAC_Close(DRV_HANDLE hMac);

TCPIP_MAC_Initialize Function

MAC driver initialization function.

File

tcpip_mac.h

C

```
SYS_MODULE_OBJ TCPIP_MAC_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- Valid handle to a driver object - if successful
- SYS_MODULE_OBJ_INVALID - if initialization failed

Description

MAC Initialize function. SYS_MODULE_OBJ TCPIP_MAC_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);

This is the function that initializes the MAC. It is called by the stack as a result of one interface going up.

Remarks

If this function fails, the stack won't turn up that interface. If the operation needs to wait for the hardware, the initialization function can return a pending code.

The returned object must be passed as argument to TCPIP_MAC_Reinitialize, TCPIP_MAC_Deinitialize, TCPIP_MAC_Tasks and TCPIP_MAC_Status routines.

Preconditions

None.

Parameters

Parameters	Description
index	Index for the MAC driver instance to be initialized

init	pointer to TCPIP_MAC_INIT initialization data containing: • macControl - Stack prepared data. • moduleData - Driver specific. Dependent on the MAC type. For PIC32 MAC driver, the TCPIP_MODULE_MAC_PIC32INT_CONFIG is used.

TCPIP_MAC_Deinitialize Function

MAC driver deinitialization function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

MAC De-Initialize function. void TCPIP_MAC_Deinitialize(SYS_MODULE_OBJ object);

This is the function that deinitializes the MAC. It is called by the stack as a result of one interface going down.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
object	Driver object handle, returned from TCPIP_MAC_Initialize

TCPIP_MAC_Reinitialize Function

MAC driver reinitialization function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

MAC Re-Initialize function. void TCPIP_MAC_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);

This is the function that re-initializes the MAC. It is called by the stack as a result of the system changing power modes. Allows reinitialization of the MAC with different power modes, etc.

Remarks

This function is optional and is not currently implemented.

Preconditions

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
object	Driver object handle, returned from the TCPIP_MAC_Initialize function

init	pointer to TCPIP_MAC_INIT initialization data containing: • macControl - Stack prepared data. • moduleData - Driver specific. Dependent on the MAC type. For PIC32 MAC driver, the TCPIP_MODULE_MAC_PIC32INT_CONFIG is used.

b) Client Control and Status Functions

TCPIP_MAC_ParametersGet Function

MAC parameter get function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_OK - if pMacParams updated properly
- TCPIP_MAC_RES error code - if processing failed for some reason

Description

This is a function that returns the run time parameters of the MAC driver.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pMacParams	address to store the MAC parameters

Function

TCPIP_MAC_RES TCPIP_MAC_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);

TCPIP_MAC_StatisticsGet Function

Gets the current MAC statistics.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_StatisticsGet(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

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRxStatistics	pointer to a TCPIP_MAC_RX_STATISTICS that will receive the current RX statistics counters. Can be NULL if not needed.
pTxStatistics	pointer to a TCPIP_MAC_TX_STATISTICS that will receive the current TX statistics counters. Can be NULL if not needed.

Function

TCPIP_MAC_RES TCPIP_MAC_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics, TCPIP_MAC_TX_STATISTICS* pTxStatistics);

TCPIP_MAC_LinkCheck Function

MAC link checking function.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is up
- false - if the link is down

Description

This is a function that returns the current status of the link.

Remarks

The stack will call this function periodically. Therefore this function is not required to initiate a new PHY transaction and to wait for the result. It can just initiate a new PHY transaction and return immediately the result of the previous transaction.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

bool TCPIP_MAC_LinkCheck(DRV_HANDLE hMac);

TCPIP_MAC_Status Function

Provides the current status of the MAC driver module.

File

tcpip_mac.h

C

```
SYS_STATUS TCPIP_MAC_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 MAC 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 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. 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 TCPIP_MAC_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from TCPIP_MAC_Initialize
SYS_STATUS          status;

status = TCPIP_MAC_Status(object);
if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from TCPIP_MAC_Initialize

Function

SYS_STATUS TCPIP_MAC_Status (SYS_MODULE_OBJ object)

TCPIP_MAC_Tasks Function

Maintains the MAC driver's state machine.

File

tcpip_mac.h

C

```
void TCPIP_MAC_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 TCPIP_MAC_Initialize routine must have been called for the specified MAC driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from TCPIP_MAC_Initialize)

Function

void TCPIP_MAC_Tasks(SYS_MODULE_OBJ object)

TCPIP_MAC_Process Function

MAC periodic processing function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK
- 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 using the TCPIP_MAC_PARAMETERS::processFlags.

An alternative approach is that the MAC driver uses a system service to create a timer signal that will call the TCPIP_MAC_Process on a periodic basis.

Remarks

Some of the processing that this function is intended for:

- The MAC driver can process its pending TX queues (although it could do that from within the TX ISR)
- RX buffers replenishing. If the number of packets in the RX queue falls below a specified limit, the MAC driver can 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

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

TCPIP_MAC_RES TCPIP_MAC_Process(DRV_HANDLE hMac);

TCPIP_MAC_RegisterStatisticsGet Function

Gets the current MAC hardware statistics registers.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_RegisterStatisticsGet(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 maintained by the MAC hardware.

Remarks

The reported values are info only and change dynamically.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRegStatistics	pointer to a pRegEntries that will receive the current hardware statistics registers values. Can be 0, if only the number of supported hardware registers is requested
nEntries	provides the number of TCPIP_MAC_STATISTICS_REG_ENTRY structures present in pRegEntries Can be 0, if only the number of supported hardware registers is requested The register entries structures will be filled by the driver, up to the supported hardware registers.
pHwEntries	pointer to an address to store the number of the statistics registers that the hardware supports It is updated by the driver. Can be 0 if not needed

Function

TCPIP_MAC_RES TCPIP_MAC_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

TCPIP_MAC_ConfigGet Function

Gets the current MAC driver configuration.

File

tcpip_mac.h

C

```
size_t TCPIP_MAC_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

The 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

TCPIP_MAC_Initialize must have been called to set up the driver. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
configBuff	pointer to a buffer to store the configuration. Can be NULL if not needed.
buffSize	size of the supplied buffer
pConfigSize	address to store the number of bytes needed for the storage of the MAC configuration. Can be NULL if not needed.

Function

size_t TCPIP_MAC_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);

c) Event Functions

TCPIP_MAC_EventAcknowledge Function

This function acknowledges a previously reported MAC event.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

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 MAC by calling TCPIP_MAC_EventPendingGet or have been passed to the MAC client by using the notification handler. Once the events are acknowledged they will 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). 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 the system performance.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EventAcknowledge( hMac, newMacEvents );
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
macEvents	MAC events that are acknowledged and re-enabled

Function

bool TCPIP_MAC_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);

TCPIP_MAC_EventMaskSet Function

MAC events report enable/disable function.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if operation succeeded
- false - if some error occurred and the operation failed

Description

This is a function that enables or disables the events to be reported to the 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 (or stack user) has to catch the events that are notified and process them. After that the stack should call TCPIP_MAC_EventAcknowledge() so that the events can be re-enabled.

Remarks

Multiple events can be "ORed" together.

The event notification system enables the user of the MAC and of the 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 null 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 MAC and reported through the notification handler it may be disabled until the TCPIP_MAC_EventAcknowledge is called.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EventMaskSet( hMac, TCPIP_MAC_EV_RX_OVFLOW | TCPIP_MAC_EV_RX_BUFNA, true );
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
macEvents	events that the MAC client wants to add/delete for notification
enable	if true, the events will be enabled, else disabled

Function

bool TCPIP_MAC_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);

TCPIP_MAC_EventPendingGet Function

Returns the currently pending MAC events.

File

tcpip_mac.h

C

```
TCPIP_MAC_EVENT TCPIP_MAC_EventPendingGet(DRV_HANDLE hMac);
```

Returns

The currently stack pending events.

Description

This function returns the currently MAC pending events. Multiple events will be "ORed" together as they accumulate. MAC processing is needed whenever a transmission related event is present (TCPIP_MAC_EV_RX_PKTPEND, TCPIP_MAC_EV_TX_DONE). Other, non critical events, may be passed to an user for informational purposes. All events have to be eventually acknowledged if re-enabling is needed.

Remarks

This is the preferred method to get the current pending MAC events. 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 returned value is just a momentary value. The pending events can change any time.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EVENT currEvents = TCPIP_MAC_EventPendingGet( hMac);
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

TCPIP_MAC_EVENT TCPIP_MAC_EventPendingGet(DRV_HANDLE hMac);

d) Filter Functions

TCPIP_MAC_RxFilterHashTableEntrySet Function

Sets the current MAC hash table receive filter.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK.
- TCPIP_MAC_RES error code - if function failed for some reason.

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.

The function will enable/disable the Hash Table receive filter if needed.

Remarks

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 TCPIP_MAC_RxFilterHashTableEntrySet() 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

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
DestMACAddr	destination MAC address (6 bytes) to allow through the Hash Table Filter. If DestMACAddr is set to 00-00-00-00-00-00, then the hash table will be cleared of all entries and the filter will be disabled.

Function

TCPIP_MAC_RES TCPIP_MAC_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);

e) Data Transfer Functions

TCPIP_MAC_PacketTx Function

MAC driver transmit function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK

- TCPIP_MAC_RES error code - if processing failed for some reason

Description

This is the MAC transmit function. Using this function a packet is submitted to the MAC driver for transmission.

Remarks

A success code returned from this function signals only that the packet was successfully scheduled for transmission over the interface and not that the packet was actually transmitted. An event will be triggered when the packet is transmitted.

The MAC driver has to support internal queuing. Since the TCPIP_MAC_PACKET data structure contains internal queuing members the MAC can queue the packet at no expense. Therefore a packet is to be rejected only if it's not properly formatted. Otherwise it has to be scheduled for transmission in an internal MAC queue.

Once the packet is scheduled for transmission the MAC driver has to 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 has to be cleared, the proper packet acknowledgment result (ackRes) has to be set and the packet acknowledgment function (ackFunc) has to be called. It is implementation dependent if all these steps are implemented as part of the ackFunc itself or as discrete steps.

On 32-bit machines, the 1st segment payload of a packet is allocated so that it is always 32-bit aligned and its size is 32 bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

PIC32 MAC driver specific : the driver checks that the segLoadOffset >= 2. See notes for the segLoadOffset member.

The packet is not required to contain the Frame Check Sequence (FCS/CRC32) field. The MAC driver/controller will insert that field itself, if it's required.

The MAC driver is required to support the transmission of multiple chained packets.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	MAC client handle obtained by a call to TCPIP_MAC_Open.
ptrPacket	pointer to a TCPIP_MAC_PACKET that's completely formatted and ready to be transmitted over the network

Function

TCPIP_MAC_RES TCPIP_MAC_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);

TCPIP_MAC_PacketRx Function

A packet is returned if such a pending packet exists.

File

tcpip_mac.h

C

```
TCPIP_MAC_PACKET* TCPIP_MAC_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const TCPIP_MAC_PACKET_RX_STAT**
ppPktStat);
```

Returns

- Valid pointer to an available RX packet
- 0 if no packet pending/available

Description

This is the MAC receive function.

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 have to be updated by the MAC driver: TCPIP_MAC_PKT_FLAG_RX has to be set. If the MAC supports it, it should set:

- TCPIP_MAC_PKT_FLAG_UNICAST has to be set if that packet is a unicast packet
- TCPIP_MAC_PKT_FLAG_BCAST has to be set if that packet is a broadcast packet
- TCPIP_MAC_PKT_FLAG_MCAST has to be set if that packet is a multicast packet
- TCPIP_MAC_PKT_FLAG_QUEUED has to be set
- TCPIP_MAC_PKT_FLAG_SPLIT has to be set if the packet has multiple data segments

Additional information about the packet is available by providing the pRes and ppPktStat fields.

Remarks

The MAC driver should dequeue and return to the caller just one single packet, and not multiple chained packets!

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 owner of that packet that it can resume control of that packet.

Once the stack modules are done processing the RX packets and the acknowledge function is called it is up to the driver design to reuse the RX packets, or simply return them to the pool they were allocated from (assuming that some sort of allocation is implemented). This document makes no requirement about how the MAC RX packets are obtained, using dynamic or static allocation techniques. This is up to the design of the MAC.

The MAC driver can use the TCPIP_MAC_Process() for obtaining new RX packets if needed.

Not all the MACs have hardware support for the received packet status. If the MAC driver cannot supply the TCPIP_MAC_PACKET_RX_STAT info, it should set the ppPktStat to 0.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRes	optional pointer to an address that will receive an additional result associated with the operation. Can be 0 if not needed.
ppPktStat	optional pointer to an address that will receive the received packet status. Note that this pointer cannot be used once the packet acknowledgment function was called. Can be 0 if not needed.

Function

```
TCPIP_MAC_PACKET*   TCPIP_MAC_PacketRx (DRV_HANDLE hMac, TCPIP_MAC_RES* pRes,
const                TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

f) Data Types and Constants

TCPIP_MAC_ACTION Enumeration

Network interface action for initialization/deinitialization

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_ACTION_INIT,
  TCPIP_MAC_ACTION_REINIT,
  TCPIP_MAC_ACTION_DEINIT,
  TCPIP_MAC_ACTION_IF_UP,
  TCPIP_MAC_ACTION_IF_DOWN
} TCPIP_MAC_ACTION;
```

Members

Members	Description
TCPIP_MAC_ACTION_INIT	stack is initialized
TCPIP_MAC_ACTION_REINIT	stack is reinitialized
TCPIP_MAC_ACTION_DEINIT	stack is deinitialized
TCPIP_MAC_ACTION_IF_UP	interface is brought up
TCPIP_MAC_ACTION_IF_DOWN	interface is brought down

Description

TCP/IP MAC Action

This enumeration defines network interface action for initialization and deinitialization.

Remarks

None.

TCPIP_MAC_ADDR Structure

Definition of a MAC address.

File

tcpip_mac.h

C

```
typedef struct {
  uint8_t v[6];
} TCPIP_MAC_ADDR;
```

Description

MAC Address

This structure defines the physical MAC address of an interface.

Remarks

None.

TCPIP_MAC_DATA_SEGMENT Structure

A data segment that's part of a TX/RX packet.

File

tcpip_mac.h

C

```
typedef struct _tag_MAC_DATA_SEGMENT {
  struct _tag_MAC_DATA_SEGMENT* next;
  uint8_t* segLoad;
  uint16_t segLen;
  uint16_t segSize;
  uint16_t segFlags;
  uint16_t segLoadOffset;
  uint8_t segClientData[4];
  uint8_t segClientLoad[0];
} TCPIP_MAC_DATA_SEGMENT;
```

Members

Members	Description
struct _tag_MAC_DATA_SEGMENT* next;	Multi-segment support, next segment in the chain.
uint8_t* segLoad;	Pointer to segment data payload. It specifies the address of the 1st byte to be transmitted.
uint16_t segLen;	Segment payload size; Number of bytes from this segment that has to be transmitted.
uint16_t segSize;	Segment allocated total usable size. This does not include the segLoadOffset (see below).
uint16_t segFlags;	TCPIP_MAC_SEGMENT_FLAGS segment flags: TCPIP_MAC_SEG_FLAG_STATIC, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX_STICKY
uint16_t segLoadOffset;	Note 1: This offset is used as a performance improvement. It allows for the MAC frame to start on an unaligned address but enforces the alignment of the network layer data and improves the IP checksum calculation. The value of this offset is MAC dependent. A typical value for an Ethernet MAC should be 2 bytes (size of the MAC frame is 14 bytes). Note 2: Normally only the 1st segment of a packet needs this extra room at the beginning of the segment buffer. Note 3: The MAC may make use of this space at the beginning of the segment buffer. This is a space reserved for the MAC purposes. Note 4: It is up to the MAC to check that the value of this offset is enforced. PIC32 MAC specific notes: Note 1. The MAC will reject the packet if the load offset is not at least 2 bytes. 2. The PIC32 MAC uses these 2 bytes to calculate the offset between the segLoad and the TCPIP_MAC_PACKET packet it belongs to. That means that the TCPIP_MAC_PACKET* and the segLoad cannot be more than 64 KB apart!
uint8_t segClientData[4];	Additional client segment data. Ignored by the MAC driver.
uint8_t segClientLoad[0];	Additional client segment payload; Ignored by the MAC driver.

Structures

	Name	Description
	_tag_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.

Description

TCPIP MAC Data Segment

Structure of a segment buffer transferred with the MAC. A MAC TX or RX packet can consist of multiple data segments. On TX the MAC has to be able to transmit packets that span multiple data segments. On RX of a network frame the MAC may have to use multiple segments to construct a packet. (For performance reasons, a contiguous MAC packet, with just one segment, if possible, is preferred).

Remarks

See notes for the segLoadOffset member. On 32-bit machines, the segment payload is allocated so that it is always 32-bit aligned and its size is 32-bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

_tag_MAC_DATA_SEGMENT Structure

A data segment that's part of a TX/RX packet.

File

tcpip_mac.h

C

```
struct _tag_MAC_DATA_SEGMENT {
  struct _tag_MAC_DATA_SEGMENT* next;
  uint8_t* segLoad;
  uint16_t segLen;
  uint16_t segSize;
  uint16_t segFlags;
  uint16_t segLoadOffset;
  uint8_t segClientData[4];
  uint8_t segClientLoad[0];
};
```

Members

Members	Description
struct _tag_MAC_DATA_SEGMENT* next;	Multi-segment support, next segment in the chain.
uint8_t* segLoad;	Pointer to segment data payload. It specifies the address of the 1st byte to be transmitted.
uint16_t segLen;	Segment payload size; Number of bytes from this segment that has to be transmitted.
uint16_t segSize;	Segment allocated total usable size. This does not include the segLoadOffset (see below).
uint16_t segFlags;	TCPIP_MAC_SEGMENT_FLAGS segment flags: TCPIP_MAC_SEG_FLAG_STATIC, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX_STICKY
uint16_t segLoadOffset;	Note 1: This offset is used as a performance improvement. It allows for the MAC frame to start on an unaligned address but enforces the alignment of the network layer data and improves the IP checksum calculation. The value of this offset is MAC dependent. A typical value for an Ethernet MAC should be 2 bytes (size of the MAC frame is 14 bytes). Note 2: Normally only the 1st segment of a packet needs this extra room at the beginning of the segment buffer. Note 3: The MAC may make use of this space at the beginning of the segment buffer. This is a space reserved for the MAC purposes. Note 4: It is up to the MAC to check that the value of this offset is enforced. PIC32 MAC specific notes: Note 1. The MAC will reject the packet if the load offset is not at least 2 bytes. 2. The PIC32 MAC uses these 2 bytes to calculate the offset between the segLoad and the TCPIP_MAC_PACKET packet it belongs to. That means that the TCPIP_MAC_PACKET* and the segLoad cannot be more than 64 KB apart!
uint8_t segClientData[4];	Additional client segment data. Ignored by the MAC driver.
uint8_t segClientLoad[0];	Additional client segment payload; Ignored by the MAC driver.

Description

TCPIP MAC Data Segment

Structure of a segment buffer transferred with the MAC. A MAC TX or RX packet can consist of multiple data segments. On TX the MAC has to be able to transmit packets that span multiple data segments. On RX of a network frame the MAC may have to use multiple segments to construct a packet. (For performance reasons, a contiguous MAC packet, with just one segment, if possible, is preferred).

Remarks

See notes for the segLoadOffset member. On 32-bit machines, the segment payload is allocated so that it is always 32-bit aligned and its size is 32-bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

TCPIP_MAC_ETHERNET_HEADER Structure

Definition of a MAC frame header.

File

tcpip_mac.h

C

```
typedef struct {
  TCPIP_MAC_ADDR DestMACAddr;
  TCPIP_MAC_ADDR SourceMACAddr;
  uint16_t Type;
} TCPIP_MAC_ETHERNET_HEADER;
```

Description

MAC Ethernet Header

This structure defines the generic Ethernet header that starts all the Ethernet frames.

Remarks

None.

TCPIP_MAC_EVENT Enumeration

Defines the possible MAC event types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_EV_NONE = 0x0000,
  TCPIP_MAC_EV_RX_PKTPEND = 0x0001,
  TCPIP_MAC_EV_RX_OVFLOW = 0x0002,
  TCPIP_MAC_EV_RX_BUFNA = 0x0004,
  TCPIP_MAC_EV_RX_ACT = 0x0008,
  TCPIP_MAC_EV_RX_DONE = 0x0010,
  TCPIP_MAC_EV_RX_FWMARK = 0x0020,
  TCPIP_MAC_EV_RX_EWMARK = 0x0040,
  TCPIP_MAC_EV_RX_BUSERR = 0x0080,
  TCPIP_MAC_EV_TX_DONE = 0x0100,
  TCPIP_MAC_EV_TX_ABORT = 0x0200,
  TCPIP_MAC_EV_TX_BUSERR = 0x0400,
  TCPIP_MAC_EV_CONN_ESTABLISHED = 0x0800,
  TCPIP_MAC_EV_CONN_LOST = 0x1000,
  TCPIP_MAC_EV_RX_ALL = (TCPIP_MAC_EV_RX_PKTPEND|TCPIP_MAC_EV_RX_OVFLOW|
TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_ACT| TCPIP_MAC_EV_RX_DONE|TCPIP_MAC_EV_RX_FWMARK|
TCPIP_MAC_EV_RX_EWMARK|TCPIP_MAC_EV_RX_BUSERR),
  TCPIP_MAC_EV_TX_ALL = (TCPIP_MAC_EV_TX_DONE|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR),
  TCPIP_MAC_EV_RXTX_ERRORS = (TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA|
TCPIP_MAC_EV_RX_BUSERR|TCPIP_MAC_EV_TX_ABORT| TCPIP_MAC_EV_TX_BUSERR),
  TCPIP_MAC_EV_CONN_ALL = (TCPIP_MAC_EV_CONN_ESTABLISHED|TCPIP_MAC_EV_CONN_LOST)
} TCPIP_MAC_EVENT;
```

Members

Members	Description
TCPIP_MAC_EV_NONE = 0x0000	no event
TCPIP_MAC_EV_RX_PKTPEND = 0x0001	RX triggered events: A receive packet is pending
TCPIP_MAC_EV_RX_OVFLOW = 0x0002	RX triggered events: RX FIFO overflow (system level latency, no descriptors, etc.)
TCPIP_MAC_EV_RX_BUFNA = 0x0004	RX triggered events: no RX descriptor available to receive a new packet

TCPIP_MAC_EV_RX_ACT = 0x0008	RX triggered events: There's RX data available
TCPIP_MAC_EV_RX_DONE = 0x0010	RX triggered events: A packet was successfully received
TCPIP_MAC_EV_RX_FWMARK = 0x0020	RX triggered events: the number of received packets is greater than or equal to than the RX Full Watermark
TCPIP_MAC_EV_RX_EWMARK = 0x0040	RX triggered events: the number of received packets is less than or equal to than the RX Empty Watermark
TCPIP_MAC_EV_RX_BUSERR = 0x0080	RX triggered events: a bus error encountered during an RX transfer
TCPIP_MAC_EV_TX_DONE = 0x0100	TX triggered events: A packet was transmitted and its status is available
TCPIP_MAC_EV_TX_ABORT = 0x0200	TX triggered events: a TX packet was aborted by the MAC (jumbo/system underrun/excessive defer/late collision/excessive collisions)
TCPIP_MAC_EV_TX_BUSERR = 0x0400	TX triggered events: a bus error encountered during a TX transfer
TCPIP_MAC_EV_CONN_ESTABLISHED = 0x0800	Connection triggered events: Connection established
TCPIP_MAC_EV_CONN_LOST = 0x1000	Connection triggered events: Connection lost
TCPIP_MAC_EV_RX_ALL = (TCPIP_MAC_EV_RX_PKTPEND|TCPIP_MAC_EV_RX_OVFLOW| TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_ACT| TCPIP_MAC_EV_RX_DONE|TCPIP_MAC_EV_RX_FWMARK| TCPIP_MAC_EV_RX_EWMARK|TCPIP_MAC_EV_RX_BUSERR)	Useful Masks: all RX related events
TCPIP_MAC_EV_TX_ALL = (TCPIP_MAC_EV_TX_DONE|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR)	Useful Masks: all TX related events
TCPIP_MAC_EV_RXTX_ERRORS = (TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA| TCPIP_MAC_EV_RX_BUSERR|TCPIP_MAC_EV_TX_ABORT| TCPIP_MAC_EV_TX_BUSERR)	abnormal traffic/system events: Action should be taken accordingly by the stack (or the stack user)
TCPIP_MAC_EV_CONN_ALL = (TCPIP_MAC_EV_CONN_ESTABLISHED|TCPIP_MAC_EV_CONN_LOST)	Mask of all Connection related events

Description

TCP/IP MAC Event

TCP/IP MAC Events Codes.

This enumeration defines all the possible events that can be reported by the MAC to the stack.

Depending on the type of the hardware Ethernet/Wi-Fi interface, etc., not all events are possible.

Remarks

None.

TCPIP_MAC_EventF Type

MAC event notification handler.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_EventF)(TCPIP_MAC_EVENT event, const void* eventParam);
```

Returns

None

Description

Event notification Function: typedef void (*TCPIP_MAC_EventF)(TCPIP_MAC_EVENT event, const void* eventParam);

This function describes the MAC event notification handler. This is a handler specified by the user of the MAC (the TCP/IP stack). The stack can use the handler to be notified of MAC events. Whenever a notification occurs the passed events have to be eventually processed:

- Stack should process the TCPIP_EV_RX_PKTPEND/TCPIP_EV_RX_DONE, TCPIP_EV_TX_DONE events
- Process the specific (error) condition

- Acknowledge the events by calling TCPIP_MAC_EventAcknowledge() so that they can be re-enabled.

Remarks

The notification handler will be called from the ISR which detects the corresponding event. The event notification handler has to be kept as short as possible and non-blocking. Mainly useful for RTOS integration where this handler will wake-up a thread that waits for a MAC event to occur. The event notification system also enables the user of the TCPIP 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 at unknown moments. Without a notification handler the stack user can still call TCPIP_MAC_EventPendingGet to see if processing by the stack needed. This is a default way of adding MAC interrupt processing to the TCP/IP stack.

Parameters

Parameters	Description
event	event that's reported (multiple events can be OR-ed)
eventParam	user parameter that's used in the notification handler

TCPIP_MAC_HANDLE Type

Handle to a MAC driver.

File

tcpip_mac.h

C

```
typedef DRV_HANDLE TCPIP_MAC_HANDLE;
```

Description

MAC Handle

This data type defines a MAC client handle.

Remarks

Another name for the DRV_HANDLE.

TCPIP_MAC_HEAP_CallocF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_CallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize);
```

Returns

- valid pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Memory Allocation Function: typedef void* (*TCPIP_MAC_HEAP_CallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize);

This calloc style function is used by the MAC to allocate memory.

Remarks

The debug version adds the module identifier and source file line number.

Parameters

Parameters	Description
heapH	heap handle to be used in call
nElems	number of identical elements requested
elemSize	size of one element, in bytes

TCPIP_MAC_HEAP_FreeF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef size_t (* TCPIP_MAC_HEAP_FreeF)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff);
```

Returns

- Non-zero number - if the free request succeeded
- 0 - if the free request failed

Description

Memory Allocation Function: typedef size_t (*TCPIP_MAC_HEAP_FreeF)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff);

This free style function is used by the MAC to free allocated memory.

Remarks

The debug version adds the module identifier and source file line number.

TCPIP_MAC_HEAP_HANDLE Type

A handle used for memory allocation functions.

File

tcpip_mac.h

C

```
typedef const void* TCPIP_MAC_HEAP_HANDLE;
```

Description

Handle to a heap

This is the handle that's required for accessing memory allocation functions.

Remarks

None

TCPIP_MAC_HEAP_MallocF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_MallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes);
```

Returns

- valid pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Memory Allocation Function: typedef void* (*TCPIP_MAC_HEAP_MallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes);

This malloc style function is used by the MAC to allocate memory.

Remarks

The debug version adds the module identifier and source file line number.

TCPIP_MAC_MODULE_CTRL Structure

Data structure that's passed to the MAC at the initialization time.

File

tcpip_mac.h

C

```
typedef struct {
  int nIfs;
  TCPIP_MAC_HEAP_MallocF mallocF;
  TCPIP_MAC_HEAP_CallocF callocF;
  TCPIP_MAC_HEAP_FreeF freeF;
  TCPIP_MAC_HEAP_HANDLE memH;
  TCPIP_MAC_PKT_AllocF pktAllocF;
  TCPIP_MAC_PKT_FreeF pktFreeF;
  TCPIP_MAC_PKT_AckF pktAckF;
  TCPIP_MAC_SynchReqF synchF;
  TCPIP_MAC_EventF eventF;
  const void* eventParam;
  unsigned int moduleId;
  int netIx;
  TCPIP_MAC_ACTION macAction;
  TCPIP_MAC_POWER_MODE powerMode;
  TCPIP_MAC_ADDR ifPhyAddress;
} TCPIP_MAC_MODULE_CTRL;
```

Members

Members	Description
int nIfs;	number of the interfaces supported in this session
TCPIP_MAC_HEAP_MallocF mallocF;	malloc type allocation function
TCPIP_MAC_HEAP_CallocF callocF;	calloc type allocation function
TCPIP_MAC_HEAP_FreeF freeF;	free type allocation free function
TCPIP_MAC_HEAP_HANDLE memH;	handle to be used in the stack allocation service calls
TCPIP_MAC_PKT_AllocF pktAllocF;	packet allocation function
TCPIP_MAC_PKT_FreeF pktFreeF;	packet free function
TCPIP_MAC_PKT_AckF pktAckF;	packet allocation function
TCPIP_MAC_SynchReqF synchF;	Synchronization object request function
TCPIP_MAC_EventF eventF;	Event notification function. used by the MAC for event reporting.
const void* eventParam;	Parameter to be used when the event function is called.
unsigned int moduleId;	Module identifier. Allows multiple channels/ports, etc. MAC support.
int netIx;	index of the current interface
TCPIP_MAC_ACTION macAction;	current action for the MAC/stack
TCPIP_MAC_POWER_MODE powerMode;	The power mode for this interface to go to. Valid only if stackAction == init/reinit. Ignored for deinitialize operation.
TCPIP_MAC_ADDR ifPhyAddress;	Physical address of the interface. MAC sets this field as part of the initialization function. The stack will use this data as the interface address.

Description

MAC Initialization Data

This is the data structure that the MAC user (TCP/IP stack) passes on to the MAC driver at the MAC initialization time. It contains all the data needed for the MAC to initialize itself and to start processing packets.

Remarks

Most of the data that's passed in this structure is permanent data. It is maintained by the stack for one full session i.e., across Initialize() -> DeInitialize() calls.

Some fields are module specific though (like the memory allocation handle, allocation functions, etc.) that could be different from one module to the other.

TCPIP_MAC_PACKET Type

Forward reference to a MAC packet.

File

tcpip_mac.h

C

```
typedef struct _tag_TCPIP_MAC_PACKET TCPIP_MAC_PACKET;
```

Structures

	Name	Description
	_tag_TCPIP_MAC_PACKET	A TX/RX packet descriptor.

Description

MAC Packet forward reference

Forward reference needed in the MAC packet acknowledge function.

Remarks

None.

_tag_TCPIP_MAC_PACKET Structure

A TX/RX packet descriptor.

File

tcpip_mac.h

C

```
struct _tag_TCPIP_MAC_PACKET {
  struct _tag_TCPIP_MAC_PACKET* next;
  struct _tag_TCPIP_MAC_PACKET* pkt_next;
  TCPIP_MAC_PACKET_ACK_FUNC ackFunc;
  const void* ackParam;
  TCPIP_MAC_DATA_SEGMENT* pDSeg;
  uint8_t* pMacLayer;
  uint8_t* pNetLayer;
  uint8_t* pTransportLayer;
  uint16_t totTransportLen;
  uint16_t pktFlags;
  uint32_t tStamp;
  const void* pktIf;
  int16_t ackRes;
  uint16_t pktClientData;
  uint32_t pktClientLoad[0];
};
```

Members

Members	Description
struct _tag_TCPIP_MAC_PACKET* next;	Multi-packet/queuing support. This field is used for chaining/queuing packets.
struct _tag_TCPIP_MAC_PACKET* pkt_next;	Multi-packet/queuing support. This field is used for chaining packet fragments, etc.
TCPIP_MAC_PACKET_ACK_FUNC ackFunc;	Packet acknowledgment function. On TX: A stack module owns the packet. Once the MAC is done transmitting the packet (success or failure) it has to set an appropriate acknowledge result in the ackRes field (a TCPIP_MAC_PKT_ACK_RES) adjust some packet flags (see TCPIP_MAC_PKT_FLAG_QUEUED) and call the packet acknowledge function (ackFunc). This call informs the packet owner that the MAC is done with using that packet. It is up to the implementation what action the ackFunc takes: reuse, free, discard the packet or if some of the above steps are included in the ackFunc itself. On RX: The packet is under the MAC control and it's passed to the stack when it contains valid data. Once the recipient of the packet (IP, ICMP, UDP, TCP, etc.) is done with the packet processing it has to set an appropriate acknowledge result in the ackRes field (a TCPIP_MAC_PKT_ACK_RES) adjust some packet flags (see TCPIP_MAC_PKT_FLAG_QUEUED) and call the packet acknowledge function (ackFunc). This call informs the packet owner (the MAC) that the processing of the packet is completed. It is up to the implementation what action the ackFunc takes: reuse, free, discard the packet or if some of the above steps are included in the ackFunc itself.
const void* ackParam;	Associated acknowledgment parameter to be used when ackFunc is called. For TX packets: this is a client supplied parameter and is not used by the MAC driver. For RX: If the MAC owns the RX packet then the MAC driver can use this field for further dispatching in the MAC driver owned acknowledgment function.

TCPIP_MAC_DATA_SEGMENT* pDSeg;	Data (root) segment associated to this packet. It is up to the design if the root data segment associated with a packet is contiguous in memory with the packet itself. It can be 0 if the packet has no associated data.
uint8_t* pMacLayer;	Pointer to the MAC frame. On TX: the sending higher layer protocol updates this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function update this field automatically).
uint8_t* pNetLayer;	Pointer to the network layer data. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function updates this field automatically. But not for IPv6!).
uint8_t* pTransportLayer;	Pointer to the transport layer. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function updates this field automatically. But not for IPv6!).
uint16_t totTransportLen;	Total length of the transport layer. The MAC driver shouldn't need this field.
uint16_t pktFlags;	TCPIP_MAC_PACKET_FLAGS associated packet flags. On TX: the MAC driver has to set: TCPIP_MAC_PKT_FLAG_QUEUED - if the driver needs to queue the packet On RX: the MAC driver updates this field before handing over the packet. Flags that the MAC driver has to set/clear: TCPIP_MAC_PKT_FLAG_RX and TCPIP_MAC_PKT_FLAG_UNICAST, TCPIP_MAC_PKT_FLAG_BCAST and TCPIP_MAC_PKT_FLAG_MCAST TCPIP_MAC_PKT_FLAG_QUEUED TCPIP_MAC_PKT_FLAG_SPLIT
uint32_t tStamp;	Time stamp value. Statistics, info. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field On RX: the MAC driver updates this field before handing over the packet.
const void* pktIf;	The packet interface. On TX: the sending higher layer protocol updates this field. The MAC driver doesn't use this field. On RX: the receiving higher level protocol updates this value. The MAC driver doesn't use this field.
int16_t ackRes;	A TCPIP_MAC_PKT_ACK_RES code associated with the packet. On TX: The MAC driver sets this field when calling the packet ackFunc. On RX: The higher level protocol which is the recipient of the packet sets this field when calling the packet ackFunc.
uint16_t pktClientData;	Client/padding data; ignored by the MAC driver. It can be used by the packet client.
uint32_t pktClientLoad[0];	Additional client packet payload, variable packet data. Ignored by the MAC driver.

Description

MAC Packet

Structure of a packet transferred with the MAC. This is the structure used for both TX and RX. See the description of each individual field.

Remarks

Specific TCP/IP stack implementations might offer packet support functions to assist in driver development. (For the MCHP TCP/IP stack see tcpip_packet.h)

Since the packets may be dynamically allocated, an acknowledge function can result in data deallocation (blocking). Therefore, the acknowledgment function should NOT be called from within an interrupt context.

TCPIP_MAC_PACKET_ACK_FUNC Type

Prototype of a MAC packet acknowledge function.

File

tcpip_mac.h

C

```
typedef bool (* TCPIP_MAC_PACKET_ACK_FUNC)(TCPIP_MAC_PACKET* pkt, const void* param);
```

Returns

- true - if the packet needs the queuing flags removed (it is not deleted and still in use)
- false - if the packet does not need the queuing flags removed (either no longer exists or the flags updated)

Description

MAC Acknowledge Function

This is the prototype of the function that the MAC calls once the TX/RX packet processing is done.

Remarks

None.

TCPIP_MAC_PACKET_FLAGS Enumeration

Flags belonging to MAC packet.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PKT_FLAG_STATIC = 0x0001,
  TCPIP_MAC_PKT_FLAG_TX = 0x0002,
  TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004,
  TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008,
  TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010,
  TCPIP_MAC_PKT_FLAG_BCAST = 0x0020,
  TCPIP_MAC_PKT_FLAG_MCAST = 0x0040,
  TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070,
  TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000,
  TCPIP_MAC_PKT_FLAG_USER = 0x0100
} TCPIP_MAC_PACKET_FLAGS;
```

Members

Members	Description
TCPIP_MAC_PKT_FLAG_STATIC = 0x0001	Packet can not be dynamically deallocated. Set when the packet is allocated.
TCPIP_MAC_PKT_FLAG_TX = 0x0002	If set, it is a TX packet/segment. Otherwise, it is a RX packet.
TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004	Packet data spans multiple segments - ZC functionality. If not set then the packet has only one data segment. It is set by the MAC driver when a RX packet spans multiple data segments.
TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008	Packet data is queued somewhere, cannot be freed. The flag is set by a module processing the packet to show that the packet is in use and queued for further processing (normally the MAC driver does that). Cleared by the packet destination when the packet processing was completed.
TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010	RX flag, MAC updated. Specifies an unicast packet.
TCPIP_MAC_PKT_FLAG_BCAST = 0x0020	RX flag, MAC updated. Specifies a broadcast packet.
TCPIP_MAC_PKT_FLAG_MCAST = 0x0040	RX flag, MAC updated. Specifies an multicast packet.
TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070	Packet cast mask bits.
TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000	Packet cast mask. Specifies a packet where the MCAST/BCAST fields are not updated by the MAC RX process.
TCPIP_MAC_PKT_FLAG_USER = 0x0100	Available user flags.

Description

MAC Packet Flags

This enumeration contains the definitions of MAC packet flags: packet allocation flags and general purpose flags.

Remarks

16 bits only packet flags are supported.

TCPIP_MAC_PACKET_RX_STAT Structure

Status of a received packet.

File

tcpip_mac.h

C

```
typedef struct {
  unsigned chksumOk : 1;
  unsigned pktChecksum : 16;
  unsigned runtPkt : 1;
  unsigned notMeUcast : 1;
  unsigned htMatch : 1;
  unsigned magicMatch : 1;
  unsigned pmMatch : 1;
```

```
    unsigned uMatch : 1;
    unsigned bMatch : 1;
    unsigned mMatch : 1;
    unsigned rxBytes : 16;
    unsigned crcError : 1;
    unsigned lenError : 1;
    unsigned lenRange : 1;
    unsigned rxOk : 1;
    unsigned mcast : 1;
    unsigned bcast : 1;
    unsigned rxCtrl : 1;
    unsigned rxVLAN : 1;
  } TCPIP_MAC_PACKET_RX_STAT;
```

Members

Members	Description
unsigned chksumOk : 1;	correct checksum filled in
unsigned pktChecksum : 16;	Packet payload checksum
unsigned runtPkt : 1;	Runt packet received
unsigned notMeUcast : 1;	Unicast, not me packet,
unsigned htMatch : 1;	Hash table match
unsigned magicMatch : 1;	Magic packet match
unsigned pmMatch : 1;	Pattern match match
unsigned uMatch : 1;	Unicast match
unsigned bMatch : 1;	Broadcast match
unsigned mMatch : 1;	Multicast match
unsigned rxBytes : 16;	Received bytes
unsigned crcError : 1;	CRC error in packet
unsigned lenError : 1;	Receive length check error
unsigned lenRange : 1;	Receive length out of range
unsigned rxOk : 1;	Receive OK
unsigned mcast : 1;	Multicast packet
unsigned bcast : 1;	Broadcast packet
unsigned rxCtrl : 1;	Control frame received
unsigned rxVLAN : 1;	Received VLAN tagged frame

Description

Received Packet Status

This structure contains the status of a received packet.

Remarks

Not all the MACs have hardware support for the received packet status.

TCPIP_MAC_PKT_ACK_RES Enumeration

List of MAC return codes for a packet acknowledge function.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PKT_ACK_NONE = 0,
  TCPIP_MAC_PKT_ACK_TX_OK = 1,
  TCPIP_MAC_PKT_ACK_RX_OK = 2,
  TCPIP_MAC_PKT_ACK_LINK_DOWN = -1,
  TCPIP_MAC_PKT_ACK_NET_DOWN = -2,
  TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3,
  TCPIP_MAC_PKT_ACK_ARP_TMO = -4,
  TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5,
  TCPIP_MAC_PKT_ACK_MAC_REJECT_ERR = -6,
  TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10,
  TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11,
  TCPIP_MAC_PKT_ACK_TYPE_ERR = -12,
```

```
  TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13,
  TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14,
  TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15,
  TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16,
  TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17
} TCPIP_MAC_PKT_ACK_RES;
```

Members

Members	Description
TCPIP_MAC_PKT_ACK_NONE = 0	packet result unknown, unspecified
TCPIP_MAC_PKT_ACK_TX_OK = 1	TX success code - packet was transmitted successfully
TCPIP_MAC_PKT_ACK_RX_OK = 2	RX success code - packet was received/processed successfully
TCPIP_MAC_PKT_ACK_LINK_DOWN = -1	TX:packet was dropped because the link was down
TCPIP_MAC_PKT_ACK_NET_DOWN = -2	TX:packet was dropped because the network is down
TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3	TX:packet was dropped because the buffer type is not supported
TCPIP_MAC_PKT_ACK_ARP_TMO = -4	TX:packet was dropped because of an ARP timeout
TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5	TX:packet associated interface is down or non existent
TCPIP_MAC_PKT_ACK_MAC_REJECT_ERR = -6	TX:packet rejected by MAC
TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10	RX: packet was dropped because the checksum was incorrect
TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11	RX: packet was dropped because of wrong interface source address
TCPIP_MAC_PKT_ACK_TYPE_ERR = -12	RX: packet was dropped because the type was unknown
TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13	RX: internal packet structure error
TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14	RX: the packet protocol couldn't find a destination for it
TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15	RX: the packet too fragmented
TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16	RX: the packet destination is closing
TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17	RX: memory allocation error

Description

MAC Packet Acknowledge Result

This enumeration contains the list of MAC codes used for a packet acknowledgment.

Remarks

16 bits only acknowledge results are supported. Positive codes indicate success. Negative codes indicate a failure of some sort.

TCPIP_MAC_PKT_AckF Type

MAC packet acknowledge function prototype.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_AckF)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes);
```

Returns

None.

Description

Packet Acknowledgment Function: typedef void (*TCPIP_MAC_PKT_AckF)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes);

This function is used by the MAC to acknowledge a TCPIP_MAC_PACKET packet when the packet processing is completed.

Remarks

A TCPIP_MAC_PACKET packet always has an acknowledgment function. This function should clear the TCPIP_MAC_PKT_FLAG_QUEUED flag. The packet's ackRes is updated only if the parameter ackRes != TCPIP_MAC_PKT_ACK_NONE. The debug version adds the module identifier.

Parameters

Parameters	Description
pPkt	pointer to a valid TCPIP_MAC_PACKET packet.

ackRes	the result of the packet processing

TCPIP_MAC_PKT_AllocF Type

MAC packet allocation function prototype.

File

tcpip_mac.h

C

```
typedef TCPIP_MAC_PACKET* (* TCPIP_MAC_PKT_AllocF)(uint16_t pktLen, uint16_t segLoadLen,
TCPIP_MAC_PACKET_FLAGS flags);
```

Returns

- Valid packet pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Packet Allocation Function: typedef TCPIP_MAC_PACKET* (*TCPIP_MAC_PKT_AllocF)(uint16_t pktLen, uint16_t segLoadLen, TCPIP_MAC_PACKET_FLAGS flags);

This function is used by the MAC to allocate a TCPIP_MAC_PACKET packet.

Remarks

The returned allocated packet should always have the TCPIP_MAC_ETHERNET_HEADER added to the packet.

The debug version adds the module identifier.

Parameters

Parameters	Description
pktLen	the size of the packet (it will be 32 bits rounded up)
segLoadLen	the payload size for the segment associated to this packet. Payload is always 32 bit aligned. If 0 no segment is created/attached to the packet.
flags	packet flags

TCPIP_MAC_PKT_FreeF Type

MAC packet free function prototype.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_FreeF)(TCPIP_MAC_PACKET* pPkt);
```

Returns

None.

Description

Packet Allocation Function: typedef void (*TCPIP_MAC_PKT_FreeF)(TCPIP_MAC_PACKET* pPkt);

This function is used by the MAC to free a previously allocated TCPIP_MAC_PACKET packet.

Remarks

The function will free a previously allocated packet. However packets or segments marked with TCPIP_MAC_PKT_FLAG_STATIC/TCPIP_MAC_SEG_FLAG_STATIC are not freed.

Also note that this function does not free explicitly the external segment payload. A payload that was created contiguously when the segment was created will be automatically freed by this function.

The debug version adds the module identifier.

Parameters

Parameters	Description
pPkt	pointer to a previously allocated packet.

TCPIP_MAC_POWER_MODE Enumeration

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_POWER_NONE,
  TCPIP_MAC_POWER_FULL,
  TCPIP_MAC_POWER_LOW,
  TCPIP_MAC_POWER_DOWN
} TCPIP_MAC_POWER_MODE;
```

Members

Members	Description
TCPIP_MAC_POWER_NONE	unknown power mode
TCPIP_MAC_POWER_FULL	up and running; valid for init/reinit
TCPIP_MAC_POWER_LOW	low power mode; valid for init/reinit
TCPIP_MAC_POWER_DOWN	interface is down

Description

This is type TCPIP_MAC_POWER_MODE.

TCPIP_MAC_PROCESS_FLAGS Enumeration

List of the MAC processing flags.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000,
  TCPIP_MAC_PROCESS_FLAG_RX = 0x0001,
  TCPIP_MAC_PROCESS_FLAG_TX = 0x0002,
  TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100
} TCPIP_MAC_PROCESS_FLAGS;
```

Members

Members	Description
TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000	the stack never has to call the TCPIP_MAC_Process function
TCPIP_MAC_PROCESS_FLAG_RX = 0x0001	the stack has to call the TCPIP_MAC_Process after an RX signal
TCPIP_MAC_PROCESS_FLAG_TX = 0x0002	the stack has to call the TCPIP_MAC_Process after an TX signal
TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100	the stack has to call the TCPIP_MAC_Process after any type of signal

Description

MAC Process Flags

List of specific MAC processing flags that indicate to the user of the MAC (TCP/IP stack) the processing that is expected by the MAC driver.

Remarks

Multiple flags can be "ORed".

TCPIP_MAC_RES Enumeration

List of return codes from MAC functions.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_RES_OK = 0,
  TCPIP_MAC_RES_PENDING = 1,
  TCPIP_MAC_RES_TYPE_ERR = -1,
  TCPIP_MAC_RES_IS_BUSY = -2,
  TCPIP_MAC_RES_INIT_FAIL = -3,
  TCPIP_MAC_RES_PHY_INIT_FAIL = -4,
  TCPIP_MAC_RES_EVENT_INIT_FAIL = -5,
  TCPIP_MAC_RES_OP_ERR = -6,
  TCPIP_MAC_RES_ALLOC_ERR = -7,
  TCPIP_MAC_RES_INSTANCE_ERR = -8,
  TCPIP_MAC_RES_FRAGMENT_ERR = -9,
  TCPIP_MAC_RES_PACKET_ERR = -10,
  TCPIP_MAC_RES_QUEUE_TX_FULL = -11,
  TCPIP_MAC_RES_SYNCH_LOCK_FAIL = -12
} TCPIP_MAC_RES;
```

Members

Members	Description
TCPIP_MAC_RES_OK = 0	operation successful
TCPIP_MAC_RES_PENDING = 1	operation is pending upon some hardware resource. call again to completion
TCPIP_MAC_RES_TYPE_ERR = -1	unsupported type
TCPIP_MAC_RES_IS_BUSY = -2	device is in use
TCPIP_MAC_RES_INIT_FAIL = -3	generic initialization failure
TCPIP_MAC_RES_PHY_INIT_FAIL = -4	PHY initialization failure
TCPIP_MAC_RES_EVENT_INIT_FAIL = -5	Event system initialization failure
TCPIP_MAC_RES_OP_ERR = -6	unsupported operation
TCPIP_MAC_RES_ALLOC_ERR = -7	memory allocation error
TCPIP_MAC_RES_INSTANCE_ERR = -8	already instantiated, initialized error
TCPIP_MAC_RES_FRAGMENT_ERR = -9	too fragmented, RX buffer too small
TCPIP_MAC_RES_PACKET_ERR = -10	unsupported/corrupted packet error
TCPIP_MAC_RES_QUEUE_TX_FULL = -11	TX queue exceeded the limit
TCPIP_MAC_RES_SYNCH_LOCK_FAIL = -12	Synchronization object lock failed Could not get a lock

Description

MAC Result Enumeration

This is the list of codes that the MAC uses to specify the outcome of a MAC function.

Remarks

Benign operation results - always have positive values. Error codes - always have negative values.

TCPIP_MAC_SEGMENT_FLAGS Enumeration

Flags belonging to MAC data segment.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_SEG_FLAG_STATIC = 0x0001,
  TCPIP_MAC_SEG_FLAG_TX = 0x0002,
  TCPIP_MAC_SEG_FLAG_RX_STICKY = 0x0004,
  TCPIP_MAC_SEG_FLAG_USER_PAYLOAD = 0x0008,
  TCPIP_MAC_SEG_FLAG_USER = 0x0100
} TCPIP_MAC_SEGMENT_FLAGS;
```

Members

Members	Description
TCPIP_MAC_SEG_FLAG_STATIC = 0x0001	Segment can not be dynamically deallocated. Set when the segment is allocated
TCPIP_MAC_SEG_FLAG_TX = 0x0002	If set, it's a TX segment; otherwise, is a RX packet.

TCPIP_MAC_SEG_FLAG_RX_STICKY = 0x0004	a MAC RX dedicated/sticky segment; otherwise, a non-dedicated/float segment
TCPIP_MAC_SEG_FLAG_USER_PAYLOAD = 0x0008	Segment carrying user payload Higher level protocols (TCP, UDP, etc.) may use it
TCPIP_MAC_SEG_FLAG_USER = 0x0100	User available segment flags.

Description

MAC Segment Flags

This enumeration contains the definitions of MAC segment flags: segment allocation flags and general purpose flags.

Remarks

16 bits only segment flags are supported.

TCPIP_MODULE_MAC_MRF24W_CONFIG Structure

File

tcpip_mac.h

C

```
typedef struct {
} TCPIP_MODULE_MAC_MRF24W_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.

TCPIP_MODULE_MAC_PIC32INT_CONFIG Structure

Data that's passed to the MAC at initialization time.

File

tcpip_mac.h

C

```
typedef struct {
  uint16_t nTxDescriptors;
  uint16_t rxBuffSize;
  uint16_t nRxDescriptors;
  uint16_t nRxDedicatedBuffers;
  uint16_t nRxInitBuffers;
  uint16_t rxLowThreshold;
  uint16_t rxLowFill;
  TCPIP_ETH_OPEN_FLAGS ethFlags;
  DRV_ETHPHY_CONFIG_FLAGS phyFlags;
  uint16_t linkInitDelay;
  uint16_t phyAddress;
  ETH_MODULE_ID ethModuleId;
  const DRV_ETHPHY_OBJECT* pPhyObject;
  const DRV_ETHPHY_OBJECT_BASE* pPhyBase;
} TCPIP_MODULE_MAC_PIC32INT_CONFIG;
```

Members

Members	Description
uint16_t nTxDescriptors;	number of TX descriptors
uint16_t rxBuffSize;	size of the corresponding RX buffer
uint16_t nRxDescriptors;	Number of RX descriptors Has to be high enough to accommodate both dedicated and non-dedicated RX buffers
uint16_t nRxDedicatedBuffers;	Number of MAC dedicated RX buffers These buffers/packets are owned by the MAC and are not returned to the packet pool They are allocated at MAC initialization time using pktAllocF and freed at MAC deinitialize time using pktFreeF Could be 0, if only not dedicated buffers are needed. For best performance usually it's best to have some dedicated buffers so as to minimize the run time allocations
uint16_t nRxInitBuffers;	Number of MAC non dedicated RX buffers allocated at the MAC initialization pktAllocF Note that these buffers are allocated in addition of the nRxDedicatedBuffers Freed at run time using pktFreeF

uint16_t rxLowThreshold;	Minimum threshold for the buffer replenish process Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers that will be freed at run time using pktFreeF Setting this value to 0 disables the buffer replenishing process
uint16_t rxLowFill;	Number of RX buffers to allocate when below threshold condition is detected If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly rxLowFill buffers
TCPIP_ETH_OPEN_FLAGS ethFlags;	flags to use for the ETH connection
DRV_ETHPHY_CONFIG_FLAGS phyFlags;	PHY configuration
uint16_t linkInitDelay;	PHYs initialization delay (milliseconds) for insuring that the PHY is ready to transmit data.
uint16_t phyAddress;	PHY address, as configured on the board. All PHYs respond to address 0
ETH_MODULE_ID ethModuleId;	Ethernet module ID for this driver instance
const DRV_ETHPHY_OBJECT* pPhyObject;	Non-volatile pointer to the PHY vendor object associated with this MAC
const DRV_ETHPHY_OBJECT_BASE* pPhyBase;	Non-volatile pointer to the PHY basic object associated with this MAC

Description

MAC Initialization Data

This structure defines the MAC initialization data for the PIC32 MAC/Ethernet controller.

Remarks

- Each supported MAC has its own specific init/configuration data

TCPIP_MAC_RX_STATISTICS Structure

MAC receive statistics data gathered at run time.

File

tcpip_mac.h

C

```
typedef struct {
  int nRxOkPackets;
  int nRxPendBuffers;
  int nRxSchedBuffers;
  int nRxErrorPackets;
  int nRxFragmentErrors;
} TCPIP_MAC_RX_STATISTICS;
```

Members

Members	Description
int nRxOkPackets;	number of OK RX packets
int nRxPendBuffers;	number of unacknowledged pending RX buffers in the driver queues. If each incoming packet fits within a RX buffer (the RX buffer is large enough) than this corresponds to the number of unacknowledged pending RX packets. Otherwise the number of packets is less than the pending buffers.
int nRxSchedBuffers;	number of currently scheduled RX buffers in the driver queues. These are available buffers, ready to receive data
int nRxErrorPackets;	number of RX packets with errors
int nRxFragmentErrors;	number of RX fragmentation errors

Description

MAC RX Statistics Data

This structure defines the RX statistics data for the MAC driver.

Remarks

This statistics are recorded by the driver, not by the hardware.

TCPIP_MAC_TX_STATISTICS Structure

MAC transmit statistics data gathered at run time.

File

tcpip_mac.h

C

```
typedef struct {
  int nTxOkPackets;
  int nTxPendBuffers;
  int nTxErrorPackets;
  int nTxQueueFull;
} TCPIP_MAC_TX_STATISTICS;
```

Members

Members	Description
int nTxOkPackets;	number of OK transmitted packets
int nTxPendBuffers;	number of unacknowledged pending TX buffers in the driver queues. This is equal with pending TX packets when each packet is contained within a TX buffer.
int nTxErrorPackets;	number of packets that could not be transmitted
int nTxQueueFull;	number of times the TX queue was full this may signal that the number of TX descriptors is too small

Description

MAC TX Statistics Data

This structure defines the TX statistics data for the MAC driver.

Remarks

This statistics are recorded by the driver, not by the hardware.

TCPIP_MAC_INIT Structure

Contains all the data necessary to initialize the MAC device.

File

tcpip_mac.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  const TCPIP_MAC_MODULE_CTRL* const macControl;
  const void* const moduleData;
} TCPIP_MAC_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
const TCPIP_MAC_MODULE_CTRL* const macControl;	Stack prepared data
const void* const moduleData;	Driver specific initialization data

Description

MAC Device Driver Initialization Data

This data structure contains all the data necessary to initialize the MAC device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the TCPIP_MAC_Initialize routine.

TCPIP_MAC_PARAMETERS Structure

Data structure that tells the MAC run time parameters.

File

tcpip_mac.h

C

```
typedef struct {
  TCPIP_MAC_ADDR ifPhyAddress;
  TCPIP_MAC_PROCESS_FLAGS processFlags;
  TCPIP_MAC_TYPE macType;
  TCPIP_MAC_LINK_MTU linkMtu;
} TCPIP_MAC_PARAMETERS;
```

Members

Members	Description
TCPIP_MAC_ADDR ifPhyAddress;	Physical address of the interface. MAC sets this field as part of the initialization process.
TCPIP_MAC_PROCESS_FLAGS processFlags;	MAC process flags. The stack will use this value to call into the MAC process function after receiving a MAC event.
TCPIP_MAC_TYPE macType;	MAC type: ETH, Wi-Fi, etc.
TCPIP_MAC_LINK_MTU linkMtu;	MAC link MTU size

Description

MAC Run time parameters

This is the data structure that the MAC user (TCP/IP stack) passes on to the MAC driver after the MAC initialization time to retrieve the settings resulted.

Remarks

None.

TCPIP_MAC_PKT_AckFDbg Type

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_AckFDbg)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_AckFDbg.

TCPIP_MAC_PKT_AllocFDbg Type

File

tcpip_mac.h

C

```
typedef TCPIP_MAC_PACKET* (* TCPIP_MAC_PKT_AllocFDbg)(uint16_t pktLen, uint16_t segLoadLen,
TCPIP_MAC_PACKET_FLAGS flags, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_AllocFDbg.

TCPIP_MAC_PKT_FreeFDbg Type

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_FreeFDbg)(TCPIP_MAC_PACKET* pPkt, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_FreeFDbg.

TCPIP_MAC_HEAP_CallocFDbg Type

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_CallocFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize,
int moduleId, int lineNo);
```

Description

This is type TCPIP_MAC_HEAP_CallocFDbg.

TCPIP_MAC_HEAP_FreeFDbg Type

File

tcpip_mac.h

C

```
typedef size_t (* TCPIP_MAC_HEAP_FreeFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff, int moduleId,
int lineNo);
```

Description

This is type TCPIP_MAC_HEAP_FreeFDbg.

TCPIP_MAC_HEAP_MallocFDbg Type

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_MallocFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes, int moduleId, int
lineNo);
```

Description

This is type TCPIP_MAC_HEAP_MallocFDbg.

TCPIP_MAC_STATISTICS_REG_ENTRY Structure

Describes a MAC hardware statistics register.

File

tcpip_mac.h

C

```
typedef struct {
  char registerName[8];
  uint32_t registerValue;
} TCPIP_MAC_STATISTICS_REG_ENTRY;
```

Members

Members	Description
char registerName[8];	name of the hardware register
uint32_t registerValue;	value of the hardware register

Description

MAC Hardware Statistics Register Entry

This structure defines an interface for gathering the MAC hardware statistics registers data.

Remarks

None

TCPIP_MAC_TYPE Enumeration

List of the MAC types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_TYPE_NONE = 0,
  TCPIP_MAC_TYPE_ETH,
  TCPIP_MAC_TYPE_WLAN,
  TCPIP_MAC_TYPES
} TCPIP_MAC_TYPE;
```

Members

Members	Description
TCPIP_MAC_TYPE_NONE = 0	invalid/unknown MAC type
TCPIP_MAC_TYPE_ETH	wired Ethernet MAC type
TCPIP_MAC_TYPE_WLAN	wireless, Wi-Fi type MAC
TCPIP_MAC_TYPES	supported types

Description

MAC Types

List of specific MAC types that indicate to the user of the MAC (TCP/IP stack) the actual type of the MAC driver.

Remarks

Other types will be eventually added.

TCPIP_MAC_SYNCH_REQUEST Enumeration

Defines the possible MAC synchronization request types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_SYNCH_REQUEST_NONE = 0,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_DELETE,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_LOCK,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_UNLOCK,
  TCPIP_MAC_SYNCH_REQUEST_CRIT_ENTER,
  TCPIP_MAC_SYNCH_REQUEST_CRIT_LEAVE
} TCPIP_MAC_SYNCH_REQUEST;
```

Members

Members	Description
TCPIP_MAC_SYNCH_REQUEST_NONE = 0	no request
TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE	request to create a synchronization object usually a binary semaphore
TCPIP_MAC_SYNCH_REQUEST_OBJ_DELETE	request to delete a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_OBJ_LOCK	request to lock access using a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_OBJ_UNLOCK	request to unlock/release access using a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_CRIT_ENTER	request to enter a critical section
TCPIP_MAC_SYNCH_REQUEST_CRIT_LEAVE	request to exit from a critical section

Description

TCP/IP MAC Synchronization object request

TCP/IP MAC synchronization request codes.

This enumeration defines all the possible synchronization actions that can be requested by the MAC to the stack at run time.

Remarks

None.

TCPIP_MAC_SynchReqF Type

MAC synchronization request function definition.

File

tcpip_mac.h

C

```
typedef bool (* TCPIP_MAC_SynchReqF)(void* synchHandle, TCPIP_MAC_SYNCH_REQUEST request);
```

Returns

- true - if the call is successful
- false - if the call failed

Description

Synchronization request Function: typedef bool (*TCPIP_MAC_SynchReqF)(void* synchHandle, TCPIP_MAC_SYNCH_REQUEST request);

This function describes the MAC synchronization request function. The MAC driver will call this function whenever it needs to create, delete, lock or unlock a synchronization object.

Remarks

None.

Parameters

Parameters	Description
synchHandle	• pointer to a valid storage area • for TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE it is an address that will store a handle to identify the synchronization object to the OS (OSAL). • for other synchronization object request types the handle has to be valid and identify the synchronization object to the OS (OSAL). • for Critical sections it maintains OS passed info
request	a TCPIP_MAC_SYNCH_REQUEST type

TCPIP_MODULE_MAC_ID Enumeration

IDs of the MAC module supported by the stack.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MODULE_MAC_NONE = 0x1000,
  TCPIP_MODULE_MAC_ENCJ60 = 0x1010,
  TCPIP_MODULE_MAC_ENCJ60_0 = 0x1010,
  TCPIP_MODULE_MAC_ENCJ600 = 0x1020,
  TCPIP_MODULE_MAC_ENCJ600_0 = 0x1020,
  TCPIP_MODULE_MAC_97J60 = 0x1030,
  TCPIP_MODULE_MAC_97J60_0 = 0x1030,
  TCPIP_MODULE_MAC_PIC32INT = 0x1040,
  TCPIP_MODULE_MAC_PIC32INT_0 = 0x1040,
  TCPIP_MODULE_MAC_MRF24W = 0x1050,
  TCPIP_MODULE_MAC_MRF24W_0 = 0x1050,
  TCPIP_MODULE_MAC_MRF24WN = 0x1060,
  TCPIP_MODULE_MAC_MRF24WN_0 = 0x1060,
```

```
  TCPIP_MODULE_MAC_EXTERNAL = 0x4000
} TCPIP_MODULE_MAC_ID;
```

Members

Members	Description
TCPIP_MODULE_MAC_NONE = 0x1000	unspecified/unknown MAC
TCPIP_MODULE_MAC_ENCJ60 = 0x1010	External ENC28J60 device: room for 16 ENCJ60 devices
TCPIP_MODULE_MAC_ENCJ60_0 = 0x1010	alternate numbered name
TCPIP_MODULE_MAC_ENCJ600 = 0x1020	External ENCX24J600 device: room for 16 ENCJ600 devices
TCPIP_MODULE_MAC_ENCJ600_0 = 0x1020	alternate numbered name
TCPIP_MODULE_MAC_97J60 = 0x1030	ETH97J60 device: room for 16 97J60 devices
TCPIP_MODULE_MAC_97J60_0 = 0x1030	alternate numbered name
TCPIP_MODULE_MAC_PIC32INT = 0x1040	internal/embedded PIC32 MAC: room for 16 PIC32 devices
TCPIP_MODULE_MAC_PIC32INT_0 = 0x1040	alternate numbered name
TCPIP_MODULE_MAC_MRF24W = 0x1050	MRF24W Wi-Fi MAC: room for 16 MRF24W devices
TCPIP_MODULE_MAC_MRF24W_0 = 0x1050	alternate numbered name
TCPIP_MODULE_MAC_MRF24WN = 0x1060	MRF24WN Wi-Fi MAC: room for 16 MRF24WN devices
TCPIP_MODULE_MAC_MRF24WN_0 = 0x1060	alternate numbered name
TCPIP_MODULE_MAC_EXTERNAL = 0x4000	External, non MCHP, MAC modules

Description

MAC Modules ID

This type defines the supported MAC types.

Remarks

Not all types listed in this enumeration are supported. New MAC types will be added as needed.

TCPIP_MODULE_MAC_MRF24WN_CONFIG Structure

File

tcpip_mac.h

C

```
typedef struct {
} TCPIP_MODULE_MAC_MRF24WN_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.

TCPIP_MAC_RX_FILTER_TYPE Enumeration

Defines the possible MAC RX filter types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_RX_FILTER_TYPE_BCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_MCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_UCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_UCAST_OTHER_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_RUNT_REJECT,
  TCPIP_MAC_RX_FILTER_TYPE_RUNT_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_REJECT,
  TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_DEFAULT
} TCPIP_MAC_RX_FILTER_TYPE;
```

Members

Members	Description
TCPIP_MAC_RX_FILTER_TYPE_BCAST_ACCEPT	Broadcast packets are accepted
TCPIP_MAC_RX_FILTER_TYPE_MCAST_ACCEPT	Multicast packets are accepted
TCPIP_MAC_RX_FILTER_TYPE_UCAST_ACCEPT	Unicast packets to a this host are accepted
TCPIP_MAC_RX_FILTER_TYPE_UCAST_OTHER_ACCEPT	Unicast packets to a different host are accepted
TCPIP_MAC_RX_FILTER_TYPE_RUNT_REJECT	Runt packets (< 64 bytes) are rejected
TCPIP_MAC_RX_FILTER_TYPE_RUNT_ACCEPT	Runt packets (< 64 bytes) are accepted
TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_REJECT	Packets with CRC errors are rejected
TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_ACCEPT	Packets with CRC errors are accepted
TCPIP_MAC_RX_FILTER_TYPE_DEFAULT	Default RX filter: will accept most valid packets

Description

MAC RX Filter Types

This enumeration defines the RX filtering capabilities for the TCP/IP MAC driver.

Remarks

The Ethernet/Wi-Fi controllers have an RX filter module that takes part in the packet acceptance decision.

Multiple values can be OR-ed together.

There usually is a priority in the RX filter processing for a MAC module. In order for a packet to be accepted, it has to be specifically accepted by a filter. Once a filter accepts or rejects a packet, further filters are not tried. If a packet isn't rejected/accepted after all filters are tried, it will be rejected by default!

Not all MACs support all the RX filter types.

Current implementation does not suport more than 16 filters.

TCPIP_MAC_LINK_MTU Enumeration

MTU size correspondig to the MAC types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_LINK_MTU_DEFAULT = 1500,
  TCPIP_MAC_LINK_MTU_ETH = 1500,
  TCPIP_MAC_LINK_MTU_WLAN = 1500
} TCPIP_MAC_LINK_MTU;
```

Members

Members	Description
TCPIP_MAC_LINK_MTU_DEFAULT = 1500	default MTU link
TCPIP_MAC_LINK_MTU_ETH = 1500	wired Ethernet MAC type
TCPIP_MAC_LINK_MTU_WLAN = 1500	wireless, Wi-Fi type MAC

Description

Link MTU per MAC Type

Lists the Maximum Transmission Unit corresponding to a MAC type.

Remarks

The MTU is usually set by the standards. For special links the values can be updated.

The default value is normally used. However the individual MAC drivers can use the specific symols

Files

Files

Name	Description
tcpip_mac.h	

tcpip_mac_config.h	Configuration file

Description

This section lists the source and header files used by the library.

tcpip_mac.h

Enumerations

	Name	Description
	TCPIP_MAC_ACTION	Network interface action for initialization/deinitialization
	TCPIP_MAC_EVENT	Defines the possible MAC event types.
	TCPIP_MAC_LINK_MTU	MTU size correspondig to the MAC types.
	TCPIP_MAC_PACKET_FLAGS	Flags belonging to MAC packet.
	TCPIP_MAC_PKT_ACK_RES	List of MAC return codes for a packet acknowledge function.
	TCPIP_MAC_POWER_MODE	This is type TCPIP_MAC_POWER_MODE.
	TCPIP_MAC_PROCESS_FLAGS	List of the MAC processing flags.
	TCPIP_MAC_RES	List of return codes from MAC functions.
	TCPIP_MAC_RX_FILTER_TYPE	Defines the possible MAC RX filter types.
	TCPIP_MAC_SEGMENT_FLAGS	Flags belonging to MAC data segment.
	TCPIP_MAC_SYNCH_REQUEST	Defines the possible MAC synchronization request types.
	TCPIP_MAC_TYPE	List of the MAC types.
	TCPIP_MODULE_MAC_ID	IDs of the MAC module supported by the stack.

Functions

	Name	Description
	TCPIP_MAC_Close	MAC driver close function.
	TCPIP_MAC_ConfigGet	Gets the current MAC driver configuration.
	TCPIP_MAC_Deinitialize	MAC driver deinitialization function.
	TCPIP_MAC_EventAcknowledge	This function acknowledges a previously reported MAC event.
	TCPIP_MAC_EventMaskSet	MAC events report enable/disable function.
	TCPIP_MAC_EventPendingGet	Returns the currently pending MAC events.
	TCPIP_MAC_Initialize	MAC driver initialization function.
	TCPIP_MAC_LinkCheck	MAC link checking function.
	TCPIP_MAC_Open	MAC driver open function.
	TCPIP_MAC_PacketRx	A packet is returned if such a pending packet exists.
	TCPIP_MAC_PacketTx	MAC driver transmit function.
	TCPIP_MAC_ParametersGet	MAC parameter get function.
	TCPIP_MAC_Process	MAC periodic processing function.
	TCPIP_MAC_RegisterStatisticsGet	Gets the current MAC hardware statistics registers.
	TCPIP_MAC_Reinitialize	MAC driver reinitialization function.
	TCPIP_MAC_RxFilterHashTableEntrySet	Sets the current MAC hash table receive filter.
	TCPIP_MAC_StatisticsGet	Gets the current MAC statistics.
	TCPIP_MAC_Status	Provides the current status of the MAC driver module.
	TCPIP_MAC_Tasks	Maintains the MAC driver's state machine.

Structures

	Name	Description
	_tag_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	_tag_TCPIP_MAC_PACKET	A TX/RX packet descriptor.
	TCPIP_MAC_ADDR	Definition of a MAC address.
	TCPIP_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	TCPIP_MAC_ETHERNET_HEADER	Definition of a MAC frame header.
	TCPIP_MAC_INIT	Contains all the data necessary to initialize the MAC device.
	TCPIP_MAC_MODULE_CTRL	Data structure that's passed to the MAC at the initialization time.
	TCPIP_MAC_PACKET_RX_STAT	Status of a received packet.

	TCPIP_MAC_PARAMETERS	Data structure that tells the MAC run time parameters.
	TCPIP_MAC_RX_STATISTICS	MAC receive statistics data gathered at run time.
	TCPIP_MAC_STATISTICS_REG_ENTRY	Describes a MAC hardware statistics register.
	TCPIP_MAC_TX_STATISTICS	MAC transmit statistics data gathered at run time.
	TCPIP_MODULE_MAC_MRF24W_CONFIG	This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.
	TCPIP_MODULE_MAC_MRF24WN_CONFIG	This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.
	TCPIP_MODULE_MAC_PIC32INT_CONFIG	Data that's passed to the MAC at initialization time.

Types

	Name	Description
	TCPIP_MAC_EventF	MAC event notification handler.
	TCPIP_MAC_HANDLE	Handle to a MAC driver.
	TCPIP_MAC_HEAP_CallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_CallocFDbg	This is type TCPIP_MAC_HEAP_CallocFDbg.
	TCPIP_MAC_HEAP_FreeF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_FreeFDbg	This is type TCPIP_MAC_HEAP_FreeFDbg.
	TCPIP_MAC_HEAP_HANDLE	A handle used for memory allocation functions.
	TCPIP_MAC_HEAP_MallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_MallocFDbg	This is type TCPIP_MAC_HEAP_MallocFDbg.
	TCPIP_MAC_PACKET	Forward reference to a MAC packet.
	TCPIP_MAC_PACKET_ACK_FUNC	Prototype of a MAC packet acknowledge function.
	TCPIP_MAC_PKT_AckF	MAC packet acknowledge function prototype.
	TCPIP_MAC_PKT_AckFDbg	This is type TCPIP_MAC_PKT_AckFDbg.
	TCPIP_MAC_PKT_AllocF	MAC packet allocation function prototype.
	TCPIP_MAC_PKT_AllocFDbg	This is type TCPIP_MAC_PKT_AllocFDbg.
	TCPIP_MAC_PKT_FreeF	MAC packet free function prototype.
	TCPIP_MAC_PKT_FreeFDbg	This is type TCPIP_MAC_PKT_FreeFDbg.
	TCPIP_MAC_SynchReqF	MAC synchronization request function definition.

Description

MAC Module Definitions for the Microchip TCP/IP Stack

File Name

tcpip_mac.h

Company

Microchip Technology Inc.

tcpip_mac_config.h

Configuration file

Macros

	Name	Description
	TCPIP_EMAC_ETH_OPEN_FLAGS	Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value
	TCPIP_EMAC_PHY_ADDRESS	The PHY address, as configured on the board. By default all the PHYs respond to address 0
	TCPIP_EMAC_PHY_CONFIG_FLAGS	Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting
	TCPIP_EMAC_PHY_LINK_INIT_DELAY	The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.

	TCPIP_EMAC_RX_BUFF_SIZE	Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DEDICATED_BUFFERS	Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DESCRIPTORS	Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers
	TCPIP_EMAC_RX_FILTERS	MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC
	TCPIP_EMAC_RX_FRAGMENTS	MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	TCPIP_EMAC_RX_INIT_BUFFERS	Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.
	TCPIP_EMAC_RX_LOW_FILL	Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers
	TCPIP_EMAC_RX_LOW_THRESHOLD	Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.
	TCPIP_EMAC_RX_MAX_FRAME	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).
	TCPIP_EMAC_TX_DESCRIPTORS	Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.

Description

MAC Configuration file

This file contains the MAC module configuration options

File Name

tcpip_mac_config.h

Company

Microchip Technology Inc.

Manager Module

This section describes the TCP/IP Stack Library Manager module.

Introduction

TCP/IP Stack Library Manager Module for Microchip Microcontrollers

This library provides the API of the Manager module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The TCP/IP manager provides access to the stack initialization, configuration and status functions.

It also process the packets received from different network interfaces (MAC drivers) and dispatches them based on their type to the proper higher layer in the stack.

Using the Library

This topic describes the basic architecture of the Manager TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_manager.h`

The interface to the Manager TCP/IP Stack library is defined in the `tcpip_manager.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Manager TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

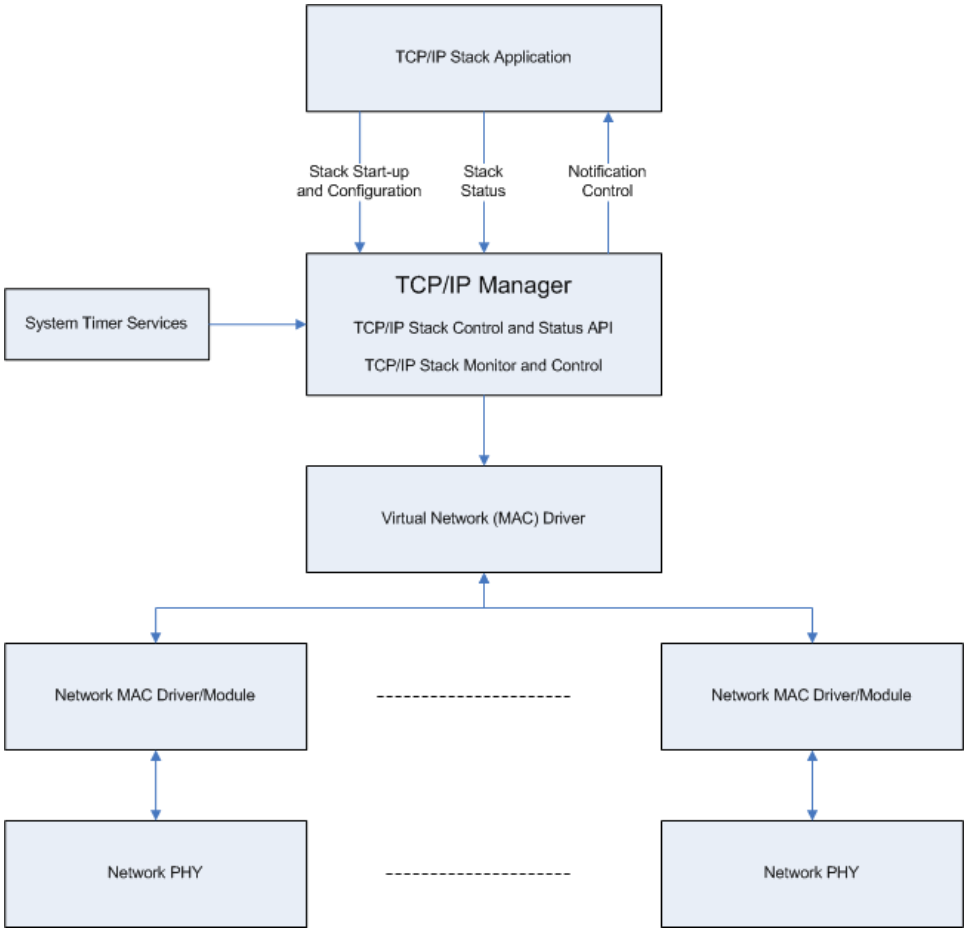
This library provides the API of the Manager TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The TCP/IP manager allows the user of the stack to initialize, control and get current status of the TCP/IP stack.

It is also responsible for the receiving of the network packets and their dispatch to the stack processing layers.

Manager 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 Manager module.

Library Interface Section	Description
Task and Initialize Functions	Routines to Initialize and Deinitialize this Module
Network Up/Down/Linked Functions	Routines to Set and Get the Network's Up/Down Status
Network Status and Control Functions	Routines to Set and Get the Various Network Controls
Network Address Status and Control Functions	Routines to Set and Get the Network's Address Status
Network Structure Storage Functions	Network Information Routines
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the TCP/IP Stack Library Manager module operates.

Description

The Manager module of the TCP/IP stack is the central module in the stack. It is the module that allows the client to configure and initialize the stack and also to get current status and parameters of the network interfaces that the stack supports.

The stack manager controls the communication to the underlying virtual MAC driver that implements the interface to a specific network channel. The various network interfaces that the TCP/IP stack supports are brought up or torn down by the stack manager in response to the stack client commands.

The manager is also responsible for providing the TCP/IP client with notifications of the events that occur in the stack at the network interface level. A dynamic registration mechanism is provided for this.

Another important functionality of the stack manager is the dispatch of the incoming network packets. The manager is notified by the underlying interface drivers of the traffic related event (transmit, receive, errors, etc.) As a response to those events the manager gets the pending receive

packets from the virtual MAC driver and, based on their type, dispatches them to the appropriate higher level protocol for further processing (IPv4, ARP, IPv6).

As part of the event notification provided by the MAC driver the TCP/IP stack manager calls back into the MAC driver when this one needs processing time and can maintain statistics counters, link status, etc.

The MPLAB Harmony framework on the system running the TCP/IP stack interfaces to the stack manager mainly by using three functions, TCPIP_STACK_Initialize and TCPIP_STACK_Deinitialize for TCP/IP stack initialization, and TCPIP_STACK_Task for periodic processing and the advance of the stack state machine.

The stack is capable of running either under an Operating System (RTOS) or in a bare-metal environment. When running in a bare-metal environment a "cooperative multi-tasking" approach is needed by the TCP/IP stack, meaning that control has to reach the TCPIP_STACK_Task function. That is the system will make periodic calls to this function which allows the TCP/IP stack to dequeue incoming packets and the corresponding state machines in the stack to advance. The application has to be aware that it cannot monopolize the processor for long periods of time, and needs to relinquish control so that the system loops perform their tasks.

In an RTOS environment it is preferred to have the stack manager run in its own thread. This thread will be blocked waiting for network interface events and when such an event occurs it will wake up and process the data. This is a more efficient way of using the CPU resources.

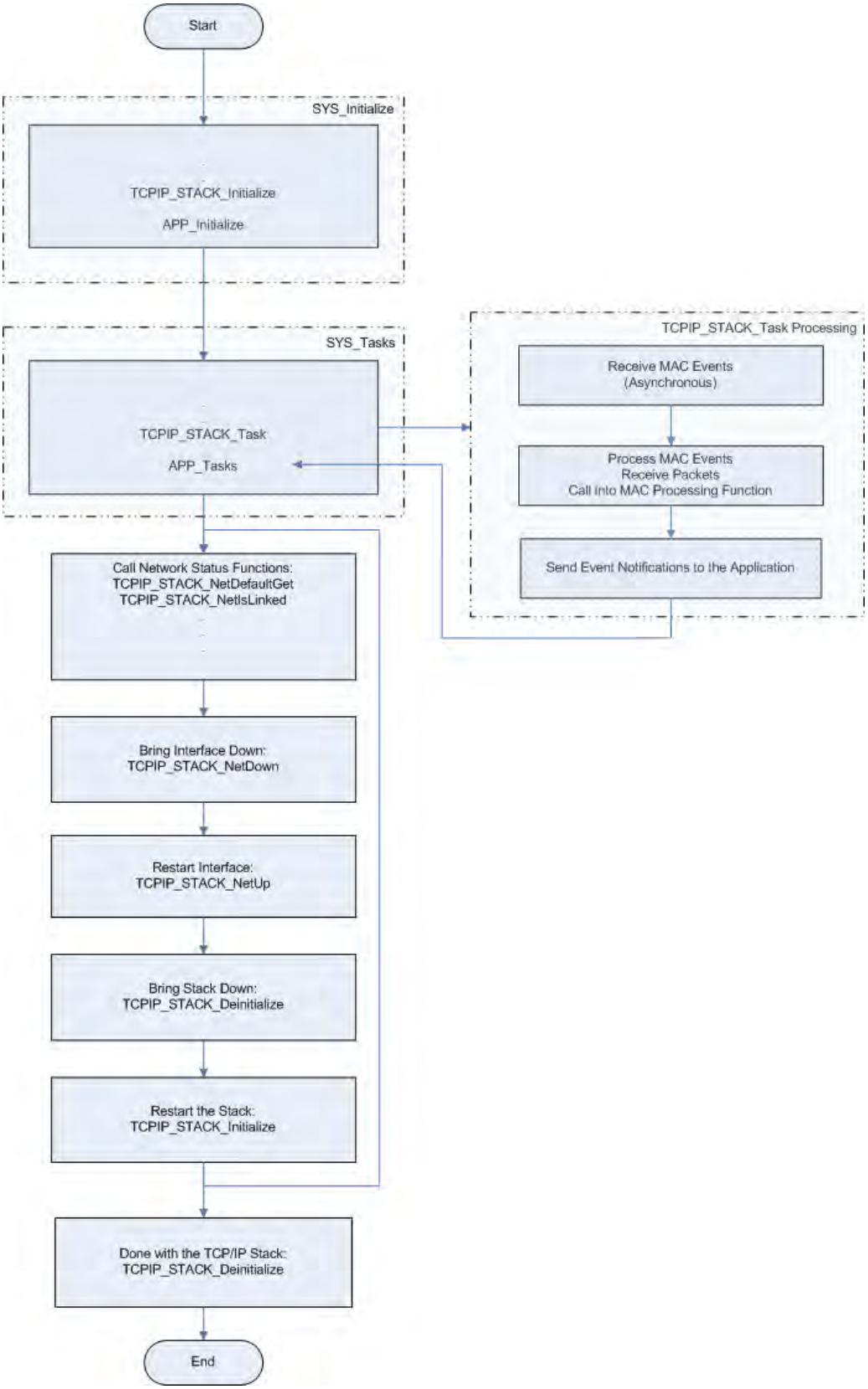
Refer to Volume IV: MPLAB Harmony Framework Reference for information on configuring the MPLAB Harmony framework to run in a RTOS or bare-metal environment.

Core Functionality

This topic describes the core functionality of the TCP/IP Stack Library Manager module.

Description

The following diagram provides an overview for the operation and use of the Manager module in a typical MPLAB Harmony framework (bare-metal, non-OS environment).



The TCP/IP stack is initialized as a part of the system initialization by a call to TCPIP_STACK_Initialize. By configuring the stack, the application can pass run-time initialization data to the stack to select both the network parameters of the interfaces (network addresses, masks, physical addresses, NetBIOS names, DHCP operation, etc.) and the parameters for the modules contained in the stack (TCP, UDP, ARP, IPv4, IPv6, etc.).

Once the stack is initialized the application can use the stack manager calls to inquire about the stack current settings (current addresses obtained by DHCP or IPv6 advertisements messages, etc.), to check the physical link status or to set new parameters to the stack.

It should be noted that different interfaces can be stopped and restarted dynamically (TCPIP_STACK_NetDown, TCPIP_STACK_NetUp) without the stack being killed. When a particular interface is brought up only the network parameters for that interface need to be supplied (no parameters

for the TCP/IP stack modules are needed).

The entire stack can be stopped and restarted (TCPIP_STACK_Deinitialize, TCPIP_STACK_Initialize). Reinitializing the stack allows the stack initialization with a complete new set of network and/or module parameters.

The stack processing is done in the TCPIP_STACK_Task function. As explained in the previous paragraph, processing time is required by the stack, meaning that periodic calls have to be made by the system/framework to the TCPIP_STACK_Task function. When execution reaches this function the stack maintains and advances its internal state machine. The most important part of this is the processing of the events that are reported by the underlying MAC drivers (through the use of the virtual MAC driver, which provides an interrupt signaling/event notification mechanism). As a response to the MAC events, the stack manager will retrieve packets from the internal MAC receive queues and dispatch them for processing. It will also call into MAC processing functions if required and provide processing for the stack modules that are the recipients of the dispatched network packets. The stack manager will also notify the application of the network events if the application dynamically registered an event notification handler with the stack.

It should be noted that the stack manager also requires a periodic tick signal for which it uses the MPLAB Harmony system timer services (see Timer System Service Library).

If the application no longer uses the TCP/IP stack, it is important that it calls the TCPIP_STACK_Deinitialize function. This function closes any open network connections (sockets, etc.), closes the underlying MAC and PHY drivers and all of the modules in the stack, and frees all of the resources that were internally allocated by the stack as part of the initialization and run-time processing.

Configuring the Library

The configuration of the Manager TCP/IP Stack is based on the file `system_config.h`, which may include `tcpip_config.h`.

This header file contains the configuration of the stack and the modules that are included in the current build of the stack.

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 Manager module of the TCP/IP Stack Library.

Description

The Manager module is part of the core functionality of the TCP/IP Stack Library. Please refer to the Building the Library topic in the TCP/IP Stack Library section for build information.

Library Interface

a) Helper Functions

	Name	Description
	TCPIP_Helper_FormatNetBIOSName	Formats a string to a valid NetBIOS name.
	TCPIP_Helper_htonl	Conversion routines from network order to host order and reverse.
	TCPIP_Helper_htons	This is function TCPIP_Helper_htons.
	TCPIP_Helper_IPAddressToString	Converts an IPV4 address to an ASCII string.
	TCPIP_Helper_IPv6AddressToString	Converts an IPv6 address to a string representation.
	TCPIP_Helper_IsBcastAddress	Checks if an IPv4 address is a broadcast address.
	TCPIP_Helper_IsMcastAddress	Checks if an IPv4 address is a multicast address.
	TCPIP_Helper_IsPrivateAddress	Detects a private (non-routable) address.
	TCPIP_Helper_MACAddressToString	Converts a MAC address to a string.
	TCPIP_Helper_ntohl	!defined(__PIC32MX__)
	TCPIP_Helper_ntohs	This is function TCPIP_Helper_ntohs.
	TCPIP_Helper_StringToIPAddress	Converts an ASCII string to an IPV4 address.
	TCPIP_Helper_StringToIPv6Address	Converts a string to an IPv6 address.
	TCPIP_Helper_StringToMACAddress	Converts a string to an MAC address.
	TCPIP_Helper_htonll	This is function TCPIP_Helper_htonll.
	TCPIP_Helper_ntohll	This is function TCPIP_Helper_ntohll.
	TCPIP_Helper_SecurePortGetByIndex	Returns the secure port belonging to a specified index.
	TCPIP_Helper_SecurePortSet	Sets the required port secure connection status.
	TCPIP_Helper_TCPSecurePortGet	Checks if the required TCP port is a secure port.
	TCPIP_Helper_UDPSecurePortGet	Checks if the required UDP port is a secure port.
	TCPIP_Helper_Base64Decode	Helper function to decode a Base-64 encoded array.
	TCPIP_Helper_Base64Encode	Helper function to encode to Base-64.

b) Task and Initialize Functions

	Name	Description
	TCPIP_STACK_SetLocalMasks	Sets the local/non-local masks for network interface.
	TCPIP_STACK_SetLocalMasksType	Sets the local/non-local masks type for network interface.
	TCPIP_STACK_HandlerDeregister	Deregisters an event notification handler.
	TCPIP_STACK_HandlerRegister	Sets a new event notification handler.
	TCPIP_STACK_Initialize	Stack initialization function.
	TCPIP_STACK_Deinitialize	Stack deinitialization function.
	TCPIP_STACK_Task	TCP/IP Stack task function.
	TCPIP_STACK_VersionGet	Gets the TCP/IP stack version in numerical format.
	TCPIP_STACK_VersionStrGet	Gets the TCP/IP stack version in string format.
	TCPIP_STACK_MACObjectGet	Returns the network MAC driver object of this interface.
	TCPIP_MODULE_SignalFunctionDeregister	Deregisters a signal function for a stack module.
	TCPIP_MODULE_SignalFunctionRegister	Registers a new signal function for a stack module.
	TCPIP_MODULE_SignalGet	Returns the current signals for a TCP/IP module.

c) Network Status and Control Functions

	Name	Description
	TCPIP_STACK_IndexToNet	Network interface handle from interface number.
	TCPIP_STACK_NetBIOSName	Network interface NetBIOS name.
	TCPIP_STACK_NetBiosNameSet	Sets network interface NetBIOS name.
	TCPIP_STACK_NetDefaultGet	Default network interface handle.
	TCPIP_STACK_NetDefaultSet	Sets the default network interface handle.
	TCPIP_STACK_NetHandleGet	Network interface handle from a name.
	TCPIP_STACK_NetIndexGet	Network interface number from interface handle.
	TCPIP_STACK_NetMask	Network interface IPv4 address mask.
	TCPIP_STACK_NetNameGet	Network interface name from a handle.
	TCPIP_STACK_NumberOfNetworksGet	Number of network interfaces in the stack.
	TCPIP_STACK_EventsPendingGet	Returns the currently pending events.
	TCPIP_STACK_NetIsReady	Gets the network interface configuration status.
	TCPIP_STACK_NetMACRegisterStatisticsGet	Get the MAC statistics register data.
	TCPIP_STACK_NetAliasNameGet	Network interface alias name from a handle.
	TCPIP_STACK_InitializeDataGet	Get the TCP/IP stack initialization data.

d) Network Up/Down/Linked Functions

	Name	Description
	TCPIP_STACK_NetIsLinked	Gets the network interface link status.
	TCPIP_STACK_NetIsUp	Gets the network interface up or down status.
	TCPIP_STACK_NetUp	Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.
	TCPIP_STACK_NetDown	Turns down a network interface.

e) Network Address Status and Control Functions

	Name	Description
	TCPIP_STACK_NetAddress	Network interface IPv4 address.
	TCPIP_STACK_NetAddressBcast	Network interface broadcast address.
	TCPIP_STACK_NetAddressDnsPrimary	Network interface DNS address.
	TCPIP_STACK_NetAddressDnsPrimarySet	Sets network interface IPv4 DNS address.
	TCPIP_STACK_NetAddressDnsSecond	Network interface secondary DNS address.
	TCPIP_STACK_NetAddressDnsSecondSet	Sets network interface IPv4 secondary DNS address.
	TCPIP_STACK_NetAddressGateway	Network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressGatewaySet	Sets network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressMac	Network interface MAC address.
	TCPIP_STACK_NetAddressMacSet	Sets network interface MAC address.
	TCPIP_STACK_NetAddressSet	Sets network interface IPv4 address.

	TCPIP_STACK_NetIPv6AddressGet	Gets network interface IPv6 address.
	TCPIP_STACK_ModuleConfigGet	Get stack module configuration data.
	TCPIP_STACK_NetMACIdGet	Get the MAC ID of the network interface.
	TCPIP_STACK_NetMACStatisticsGet	Get the MAC statistics data.
	TCPIP_STACK_Status	Provides the current status of the TCPIP stack module.

f) Network Structure Storage Functions

	Name	Description
	TCPIP_STACK_NetConfigGet	Get stack network interface configuration data.
	TCPIP_STACK_NetConfigSet	Restores stack network interface configuration data.

g) Data Types and Constants

	Name	Description
	TCPIP_LOCAL_MASK_TYPE	Mask operation for local/non-local network interface.
	TCPIP_NET_HANDLE	Defines a network interface handle.
	TCPIP_EVENT	Defines the possible TCPIP event types.
	TCPIP_EVENT_HANDLE	Defines a TCPIP stack event handle.
	TCPIP_STACK_EVENT_HANDLER	Pointer to a function(handler) that will get called to process an event.
	TCPIP_Helper_ntohl	This is macro TCPIP_Helper_ntohl.
	TCPIP_Helper_ntohs	This is macro TCPIP_Helper_ntohs.
	TCPIP_STACK_IF_NAME_ALIAS_ETH	alias for Ethernet interface
	TCPIP_STACK_IF_NAME_ALIAS_UNK	alias for unknown interface
	TCPIP_STACK_IF_NAME_ALIAS_WLAN	alias for Wi-Fi interface
	_IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IP_ADDR	Backward compatible definition to represent an IPv4 address.
	IP_ADDRESS_TYPE	Definition of the TCP/IP supported address types.
	IP_MULTI_ADDRESS	Definition to represent multiple IP addresses.
	IPV4_ADDR	Definition to represent an IPv4 address
	IPV6_ADDR	Definition to represent an IPv6 address.
	IPV6_ADDR_HANDLE	Definition to represent an IPv6 address.
	IPV6_ADDR_SCOPE	Definition to represent the scope of an IPv6 address.
	IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IPV6_ADDR_TYPE	Definition to represent the type of an IPv6 address.
	TCPIP_NETWORK_CONFIG	Defines the data required to initialize the network configuration.
	TCPIP_NETWORK_CONFIG_FLAGS	Definition of network configuration start-up flags.
	TCPIP_STACK_INIT	Definition to represent the TCP/IP stack initialization/configuration structure.
	TCPIP_STACK_MODULE	List of the TCP/IP stack supported modules.
	TCPIP_STACK_MODULE_CONFIG	Definition to represent a TCP/IP module initialization/configuration structure.
	TCPIP_MODULE_SIGNAL	Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.
	TCPIP_MODULE_SIGNAL_FUNC	Pointer to a function(handler) that will get called when a stack internal signal occurred.
	TCPIP_MODULE_SIGNAL_HANDLE	Defines a TCPIP stack signal function handle.
	TCPIP_STACK_IF_NAME_97J60	This is macro TCPIP_STACK_IF_NAME_97J60.
	TCPIP_STACK_IF_NAME_ENCJ60	This is macro TCPIP_STACK_IF_NAME_ENCJ60.
	TCPIP_STACK_IF_NAME_ENCJ600	This is macro TCPIP_STACK_IF_NAME_ENCJ600.
	TCPIP_STACK_IF_NAME_MRF24W	This is macro TCPIP_STACK_IF_NAME_MRF24W.
	TCPIP_STACK_IF_NAME_NONE	Defines the TCP/IP stack supported network interfaces.
	TCPIP_STACK_IF_NAME_PIC32INT	This is macro TCPIP_STACK_IF_NAME_PIC32INT.
	TCPIP_STACK_IF_POWER_DOWN	powered down, not started
	TCPIP_STACK_IF_POWER_FULL	up and running;
	TCPIP_STACK_IF_POWER_LOW	low power mode; not supported now
	TCPIP_STACK_IF_POWER_NONE	unsupported, invalid
	TCPIP_STACK_VERSION_MAJOR	TCP/IP stack version
	TCPIP_STACK_VERSION_MINOR	This is macro TCPIP_STACK_VERSION_MINOR.
	TCPIP_STACK_VERSION_PATCH	This is macro TCPIP_STACK_VERSION_PATCH.

	TCPIP_STACK_VERSION_STR	This is macro TCPIP_STACK_VERSION_STR.
	TCPIP_Helper_ntohll	This is macro TCPIP_Helper_ntohll.
	TCPIP_STACK_IF_NAME_MRF24WN	This is macro TCPIP_STACK_IF_NAME_MRF24WN.

Description

This section describes the Application Programming Interface (API) functions of the Manager module.

Refer to each section for a detailed description.

a) Helper Functions

TCPIP_Helper_FormatNetBIOSName Function

Formats a string to a valid NetBIOS name.

File

tcpip_helpers.h

C

```
void TCPIP_Helper_FormatNetBIOSName(uint8_t Name[]);
```

Returns

None.

Description

This function formats a string to a valid NetBIOS name. Names will be exactly 16 characters, as defined by the NetBIOS spec. The 16th character will be a 0x00 byte, while the other 15 will be the provided string, padded with spaces as necessary.

Preconditions

None.

Parameters

Parameters	Description
Name	the string to format as a NetBIOS name. This parameter must have at least 16 bytes allocated.

Function

void TCPIP_Helper_FormatNetBIOSName(uint8_t Name[])

TCPIP_Helper_htonl Function

Conversion routines from network order to host order and reverse.

File

tcpip_helpers.h

C

```
inline uint32_t TCPIP_Helper_htonl(uint32_t hLong);
```

Returns

The converted 16/32-bit quantity.

Description

These functions will convert a long or short quantity from the network order (big endian) to host order (little endian on PIC32).

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
hLong/nLong	long value (32 bits) to convert
hShort/nShort	short value (16 bits) to convert

Function

uint64_t TCPIP_Helper_htonll(uint64_t hLLong);
uint32_t TCPIP_Helper_htonl(uint32_t hLong);
uint16_t TCPIP_Helper_htons(uint16_t hShort);
uint64_t TCPIP_Helper_ntohll(uint64_t nLLong);
uint32_t TCPIP_Helper_ntohl(uint32_t nLong);
uint16_t TCPIP_Helper_ntohs(uint16_t nShort);

TCPIP_Helper_htons Function

File

tcpip_helpers.h

C

```
inline uint16_t TCPIP_Helper_htons(uint16_t hShort);
```

Description

This is function TCPIP_Helper_htons.

TCPIP_Helper_IPAddressToString Function

Converts an IPV4 address to an ASCII string.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IPAddressToString(const IPV4_ADDR* IPAddress, char* buff, size_t buffSize);
```

Returns

- true - an IP address was successfully converted
- false - supplied buffer was not large enough

Description

This function converts IPV4_ADDR to a dotted-quad decimal IP address string

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to convert
buff	buffer to store the converted dotted-quad IP address string
buffSize	buffer size

Function

bool TCPIP_Helper_IPAddressToString(const IPV4_ADDR* IPAddress, char* buff, size_t buffSize);

TCPIP_Helper_IPv6AddressToString Function

Converts an IPv6 address to a string representation.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IPv6AddressToString(const IPV6_ADDR * addr, char* buff, size_t buffSize);
```

Returns

- true - an IPv6 address was successfully converted
- false - supplied buffer was not large enough

Description

This function converts an IPV6_ADDR to a text representation of an IPv6 address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
addr	Pointer to IPV6_ADDR to convert
buff	Pointer to a buffer to store the text representation
buffSize	buffer size

Function

bool TCPIP_Helper_IPv6AddressToString (const IPV6_ADDR * addr, char* buff, size_t buffSize);

TCPIP_Helper_IsBcastAddress Function

Checks if an IPv4 address is a broadcast address.

File

tcpip_helpers.h

C

```
__inline__ bool TCPIP_Helper_IsBcastAddress(const IPV4_ADDR* IPAddress);
```

Returns

- true - if the IPv4 address is a broadcast address
- false - if the IPv4 address is not a broadcast address

Description

This function verifies if the supplied IPv4 address is a broadcast address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to check

Function

bool TCPIP_Helper_IsBcastAddress(const IPV4_ADDR* IPAddress);

TCPIP_Helper_IsMcastAddress Function

Checks if an IPv4 address is a multicast address.

File

tcpip_helpers.h

C

```
__inline__ bool TCPIP_Helper_IsMcastAddress(const IPV4_ADDR* IPAddress);
```

Returns

- true - if the IPv4 address is a multicast address
- false - if the IPv4 address is not a multicast address

Description

This function verifies if the supplied IPv4 address is a multicast address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to check

Function

bool TCPIP_Helper_IsMcastAddress(const IPV4_ADDR* IPAddress)

TCPIP_Helper_IsPrivateAddress Function

Detects a private (non-routable) address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IsPrivateAddress(uint32_t ipv4Address);
```

Returns

- true - the IPv4 address is a private address
- false - the IPv4 address is a routable address

Description

This function checks if the passed in IPv4 address is a private or a routable address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
ipv4Address	IPv4 address to check, network order

Function

bool TCPIP_Helper_IsPrivateAddress(uint32_t ipv4Address);

TCPIP_Helper_MACAddressToString Function

Converts a MAC address to a string.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_MACAddressToString(const TCPIP_MAC_ADDR* macAddr, char* buff, size_t buffSize);
```

Returns

- true - a MAC address was successfully decoded
- false - no MAC address could be found, or the format was incorrect

Description

This function will convert a MAC address to a string representation.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
macAddr	Pointer to address to convert
buff	buffer to store the string representation
buffSize	size of the buffer

Function

bool TCPIP_Helper_MACAddressToString(const TCPIP_MAC_ADDR* macAddr, char* buff, size_t buffSize);

TCPIP_Helper_ntohl Function

File

tcpip_helpers.h

C

```
uint32_t TCPIP_Helper_ntohl(uint32_t nLong);
```

Description

!defined(__PIC32MX__)

TCPIP_Helper_ntohs Function

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_ntohs(uint16_t nShort);
```

Description

This is function TCPIP_Helper_ntohs.

TCPIP_Helper_StringToIPAddress Function

Converts an ASCII string to an IPV4 address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToIPAddress(const char* str, IPV4_ADDR* IPAddress);
```

Returns

- true - an IP address was successfully decoded
- false - no IP address could be found, or the format was incorrect

Description

This function parses a dotted-quad decimal IP address string into an IPV4_ADDR struct. The output result is big-endian.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
str	Pointer to a dotted-quad IP address string
IPAddress	Pointer to IPV4_ADDR in which to store the result

Function

bool TCPIP_Helper_StringToIPAddress(const char* str, IPV4_ADDR* IPAddress);

TCPIP_Helper_StringToIPv6Address Function

Converts a string to an IPv6 address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToIPv6Address(const char * str, IPV6_ADDR * addr);
```

Description

This function parses the text representation of an IPv6 address to an IPV6_ADDR struct. The output result is big-endian.

Remarks

None.

Preconditions

None

Example

1111:2222:3333:4444:5555:6666:AAAA:FFFF 1111:2222::FFFF 1111:2222:3333:4444:5555:6666:192.168.1.20 addr - Pointer to IPV6_ADDR in which to store the result

Parameters

Parameters	Description
str	Pointer to an RFC3513, Section 2.2 text representation of an IPv6 address

Function

bool TCPIP_Helper_StringToIPv6Address (const char * str, IPV6_ADDR * addr);

TCPIP_Helper_StringToMACAddress Function

Converts a string to an MAC address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToMACAddress(const char* str, uint8_t macAddr[6]);
```

Returns

- true - a MAC address was successfully decoded
- false - no MAC address could be found, or the format was incorrect

Description

This function parses a MAC address string "aa:bb:cc:dd:ee:ff" or "aa-bb-cc-dd-ee-ff" into an hex MAC address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
str	Pointer to a colon separated MAC address string
macAddr	Pointer to buffer to store the result

Function

bool TCPIP_Helper_StringToMACAddress(const char* str, uint8_t macAddr[6]);

TCPIP_Helper_htonll Function

File

tcpip_helpers.h

C

```
uint64_t TCPIP_Helper_htonll(uint64_t hLLong);
```

Description

This is function TCPIP_Helper_htonll.

TCPIP_Helper_ntohll Function

File

tcpip_helpers.h

C

```
uint64_t TCPIP_Helper_ntohll(uint64_t nLLong);
```

Description

This is function TCPIP_Helper_ntohll.

TCPIP_Helper_SecurePortGetByIndex Function

Returns the secure port belonging to a specified index.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_SecurePortGetByIndex(int index, bool streamSocket, int* pnIndexes);
```

Returns

- a port number corresponding to the required index
- 0 if the corresponding slot is free or the port is not associated with the requested stream/datagram flag.

Description

This function returns the secure TCP/UDP port situated at the internal port table requested index. It also returns the number of indexes currently in the table.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
index	The port index to query. 0 should always be a valid index
streamSocket	if true, a stream/TCP port is queried else a datagram/UDP port is queried
pnIndexes	pointer to store the number of indexes that the table currntly has. Could be NULL if not needed.

Function

uint16_t TCPIP_Helper_SecurePortGetByIndex(int index, bool streamSocket, int* pnIndexes);

TCPIP_Helper_SecurePortSet Function

Sets the required port secure connection status.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_SecurePortSet(uint16_t port, bool streamSocket, bool isSecure);
```

Returns

- true - the port status successfuly changed
- false - the port status could not be changed (no more slots in the table, port not found, etc.)

Description

This function sets the internal data for deciding if the required port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP internally maintained table with the ports that require secure connections is manipulated with this function. This table can be queried using the TCPIP_Helper_TCPSecurePortGet() function.

Remarks

Currently there is no protection for multi-threaded accessing and modifying the entries in the secure port table. It is thus recommended that the updates to the table occur just once at initialization time and after that all threads perform read-only accesses.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
port	The TCP port to set
streamSocket	if true, a stream/TCP port is queried else a datagram/UDP port is queried
isSecure	if true, the port is set as requiring secure connection if false, the port is set as not requiring a secure connection and it will be removed from the secure ports table

Function

bool TCPIP_Helper_SecurePortSet(uint16_t port, bool streamSocket, bool isSecure);

TCPIP_Helper_TCPSecurePortGet Function

Checks if the required TCP port is a secure port.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_TCPSecurePortGet(uint16_t tcpPort);
```

Returns

- true - if the port is a secure connection port
- false - if port is not secure (the port is not found in the internal secure ports table)

Description

This function checks the internal data for detecting if the required TCP port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP stack maintains an internal table with the ports that require secure connections. This table can be maintained using the TCPIP_Helper_SecurePortSet() function.

Remarks

The TCP/IP stack populates the internal secure port table with default values as part of the stack power up procedure. The table is not re-initialized when the stack is brought down and then restarted.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
tcpPort	The TCP port to query

Function

bool TCPIP_Helper_TCPSecurePortGet(uint16_t tcpPort);

TCPIP_Helper_UDPSecurePortGet Function

Checks if the required UDP port is a secure port.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_UDPSecurePortGet(uint16_t udpPort);
```

Returns

- true - if the port is a secure connection port
- false - if port is not secure (the port is not found in the internal secure ports table)

Description

This function checks the internal data for detecting if the required UDP port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP stack maintains an internal table with the ports that require secure connections. This table can be maintained using the TCPIP_Helper_SecurePortSet() function.

Remarks

The TCP/IP stack populates the internal secure port table with default values as part of the stack power up procedure. The table is not re-initialized when the stack is brought down and then restarted.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
udpPort	The UDP port to query

Function

bool TCPIP_Helper_UDPSecurePortGet(uint16_t udpPort);

TCPIP_Helper_Base64Decode Function

Helper function to decode a Base-64 encoded array.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_Base64Decode(const uint8_t* sourceData, uint16_t sourceLen, uint8_t* destData,
uint16_t destLen);
```

Returns

Number of decoded bytes written to destData.

Description

Decodes a Base-64 array to its literal representation.

Remarks

This function will ignore invalid Base-64 characters (CR, LF, etc). If sourceData is equal to destData, the data will be converted in-place. If sourceData is not equal to destData, but the regions overlap, the behavior is undefined.

Decoded data size is 3 / 4 the size of the encoded source data.

Preconditions

None

Parameters

Parameters	Description
sourceData	Pointer to a string of Base-64 encoded data
sourceLen	Length of the Base-64 source data Maximum length that can be written to destData
destData	Pointer to write the decoded data

Function

uint16_t TCPIP_Helper_Base64Decode(const uint8_t* sourceData, uint16_t sourceLen,

uint8_t* destData, uint16_t destLen)

TCPIP_Helper_Base64Encode Function

Helper function to encode to Base-64.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_Base64Encode(const uint8_t* sourceData, uint16_t sourceLen, uint8_t* destData,
uint16_t destLen);
```

Returns

Number of encoded bytes written to destData. This will always be a multiple of 4.

Description

This function encodes a binary array to Base-64.

Remarks

Encoding cannot be performed in-place. If surceData overlaps with destData, the behavior is undefined.

The source data is padded wit 1 or 2 bytes, if needed, to make the source size a multiple of 3 bytes. Then for each 3 bytes tuple in the source 4 output bytes are generated. The output size needed is pad(sourceLen) * 4 / 3 bytes.

Preconditions

None

Parameters

Parameters	Description
sourceData	Pointer to a string of binary data
sourceLen	Length of the binary source data
destData	Pointer to write the Base-64 encoded data
destLen	Maximum length that can be written to destData

Function

uint16_t TCPIP_Helper_Base64Encode(const uint8_t* sourceData, uint16_t sourceLen,

uint8_t* destData, uint16_t destLen)

b) Task and Initialize Functions

TCPIP_STACK_SetLocalMasks Function

Sets the local/non-local masks for network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_SetLocalMasks(TCPIP_NET_HANDLE netH, uint32_t andMask, uint32_t orMask);
```

Returns

- true - if interface exists and is enabled
- false - if an error occurred

Description

This function sets the masks used in the stack decision of a destination address being a local or non-local address. These masks will be used when the corresponding mask type is set to TCPIP_LOCAL_MASK_SET.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_SetLocalMasks(netH, 0x0101a0c0, 0x0);
```

Parameters

Parameters	Description
netH	handle of the interface to use
andMask	AND mask to use for local network detection, big endian (BE) format
orType	OR mask to use for local network detection , BE format

Function

bool TCPIP_STACK_SetLocalMasks(TCPIP_NET_HANDLE netH,

uint32_t andMask, uint32_t orMask)

TCPIP_STACK_SetLocalMasksType Function

Sets the local/non-local masks type for network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_SetLocalMasksType(TCPIP_NET_HANDLE netH, TCPIP_LOCAL_MASK_TYPE andType,
TCPIP_LOCAL_MASK_TYPE orType);
```

Returns

- true - if the interface exists and is enabled
- false - if an error occurred

Description

This function sets the types of masks used in the stack decision of a destination address being a local or non-local address. For example, when a TCP connection is made, the advertised MSS usually has different values for local versus non-local networks.

To decide if a destination IP address (destAdd) is local to a network interface (having the netAdd address) the following calculation is performed: if(((destAdd & andMask) | orMask) == ((netAdd & andMask) | orMask)), the destination address is considered to be a local address; otherwise, the destination address is non-local.

This function sets the types of masks used for the AND and OR operations, as follows:

- TCPIP_LOCAL_MASK_ZERO: use the all 0's mask (0x00000000)
- TCPIP_LOCAL_MASK_ONE: use the all 1's mask (0xffffffff)
- TCPIP_LOCAL_MASK_NET: use the current network mask
- TCPIP_LOCAL_MASK_SET: use a mask that's set by the application There are operations to set a specific AND and OR masks

Using different values for the AND and OR masks an application can select various destination networks to be considered as local/non-local:

- only their own network
- any network
- no network
- specific (range of) networks
- etc.

Remarks

The default value for both AND and OR masks is set to TCPIP_LOCAL_MASK_ZERO so that any destination network is considered to be local!

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_SetLocalMasksType(netH, TCPIP_LOCAL_MASK_NET, TCPIP_LOCAL_MASK_ZERO);
```

Parameters

Parameters	Description
netH	handle of the interface to use
andType	AND type of mask to use for local network detection
orType	OR type of mask to use for local network detection

Function

bool TCPIP_STACK_SetLocalMasksType(TCPIP_NET_HANDLE netH, TCPIP_LOCAL_MASK_TYPE andType,
TCPIP_LOCAL_MASK_TYPE orType)

TCPIP_STACK_HandlerDeregister Function

Deregisters an event notification handler.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_HandlerDeregister(TCPIP_EVENT_HANDLE hEvent);
```

Returns

- true - if the operation succeeded
- false - if the operation failed

Description

This function removes an event notification handler.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_EVENT_HANDLE myHandle = TCPIP_STACK_HandlerRegister( hNet, TCPIP_EV_CONN_ALL, myEventHandler, myParam
);
// do something else
// now we're done with it
TCPIP_STACK_HandlerDeregister(myHandle);
```

Parameters

Parameters	Description
hEvent	TCPIP event handle obtained by a call to TCPIP_STACK_HandlerRegister

Function

bool TCPIP_STACK_HandlerDeregister(TCPIP_EVENT_HANDLE hEvent);

TCPIP_STACK_HandlerRegister Function

Sets a new event notification handler.

File

tcpip_manager.h

C

```
TCPIP_EVENT_HANDLE TCPIP_STACK_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_EVENT evMask,
TCPIP_STACK_EVENT_HANDLER notifyHandler, const void* notifyfParam);
```

Returns

- a valid TCPIP_EVENT_HANDLE - if the operation succeeded
- NULL - if the operation failed

Description

This function sets a new event notification handler. The caller can use the handler to be notified of stack events.

Remarks

The notification handler may be called from the ISR which detects the corresponding event. The event notification handler has to be kept as short as possible and non-blocking.

Without a notification handler the stack user can still call TCPIP_STACK_GetPending() to see if processing by the stack needed.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_EVENT_HANDLE myHandle = TCPIP_STACK_HandlerRegister( hNet, TCPIP_EV_CONN_ALL, myEventHandler, myParam
);
```

Parameters

Parameters	Description
hNet	network handle
evMask	mask of events to be notified of
notifyHandler	the event notification handler
notifyfParam	notification handler parameter

Function

TCPIP_EVENT_HANDLE TCPIP_STACK_HandlerRegister(TCPIP_NET_HANDLE hNet,
TCPIP_EVENT evMask, TCPIP_STACK_EVENT_HANDLER notifyHandler, const void* notifyfParam)

TCPIP_STACK_Initialize Function

Stack initialization function.

File

tcpip_manager.h

C

```
SYS_MODULE_OBJ TCPIP_STACK_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- Valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

The function initializes the stack. If an error occurs, the SYS_ERROR() is called and the function deinitialize itself.

Remarks

This function must be called before any of the stack or its component routines are used.

New TCPIP_NETWORK_CONFIG types could be added/removed at run time for implementations that support dynamic network interface creation.

Only one instance of the TCP/IP stack can run in the system and the index parameter is irrelevant

If this call is made after the stack is successfully initialized, the current TCP/IP stack object handle will be returned.

Preconditions

None

Parameters

Parameters	Description
index	index of the TCP/IP stack to be initialized
init	Pointer to initialization data.
It should be a TCPIP_STACK_INIT structure carrying the following data	• pNetConf - pointer to an array of TCPIP_NETWORK_CONFIG to support • nNets - number of network configurations in the array • pModConfig - pointer to an array of TCPIP_STACK_MODULE_CONFIG • nModules - number of modules to initialize

Function

SYS_MODULE_OBJ TCPIP_STACK_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init);

TCPIP_STACK_Deinitialize Function

Stack deinitialization function.

File

tcpip_manager.h

C

```
void TCPIP_STACK_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function performs the deinitialization of the TCPIP stack. All allocated resources are released.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize.

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize call

Function

void TCPIP_STACK_Deinitialize(SYS_MODULE_OBJ object)

TCPIP_STACK_Task Function

TCP/IP Stack task function.

File

tcpip_manager.h

C

```
void TCPIP_STACK_Task(SYS_MODULE_OBJ object);
```

Returns

None.

Description

TCP/IP stack execution state machine. Stack Finite-state Machine (FSM) is executed.

Remarks

This FSM checks for new incoming packets, and routes it to appropriate stack components. It also performs timed operations.

This function must be called periodically to ensure timely responses.

This function continues the stack initialization process after the TCPIP_STACK_Initialize was called. The TCPIP_STACK_Status will report SYS_STATUS_BUSY while the initialization is in progress.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize.

Parameters

Parameters	Description
object	Object handle returned by TCPIP_STACK_Initialize

Function

void TCPIP_STACK_Task(SYS_MODULE_OBJ object)

TCPIP_STACK_VersionGet Function

Gets the TCP/IP stack version in numerical format.

File

tcpip_manager.h

C

```
unsigned int TCPIP_STACK_VersionGet(const SYS_MODULE_INDEX index);
```

Returns

Current driver version in numerical format.

Description

This function gets the TCP/IP stack version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using TCPIP_STACK_VersionStrGet()

Remarks

None.

Preconditions

None.

Example

```
unsigned int version;

version = TCPIP_STACK_VersionGet( TCPIP_STACK_INDEX_1 );

if(version < 110200)
{
    // Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to get the version for

Function

unsigned int TCPIP_STACK_VersionGet (const SYS_MODULE_INDEX index)

TCPIP_STACK_VersionStrGet Function

Gets the TCP/IP stack version in string format.

File

tcpip_manager.h

C

```
char * TCPIP_STACK_VersionStrGet(const SYS_MODULE_INDEX index);
```

Returns

Current TCP/IP stack version in the string format.

Description

This function gets the TCP/IP stack version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using TCPIP_STACK_VersionGet.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the object instance to get the version for.

Function

char * TCPIP_STACK_VersionStrGet (const SYS_MODULE_INDEX index)

TCPIP_STACK_MACObjectGet Function

Returns the network MAC driver object of this interface.

File

tcpip_manager.h

C

```
const TCPIP_MAC_OBJECT* TCPIP_STACK_MACObjectGet(TCPIP_NET_HANDLE netH);
```

Returns

It returns a valid MAC driver object pointer if success. Returns 0 if no such interface or there is no MAC object.

Description

This function returns the MAC driver object that's associated with the interface handle.

Remarks

The MAC driver object is the one that's passed at the stack/interface initialization.

The MAC driver is not a true multi-client driver. Under normal circumstances the MAC driver has only one client, the TCP/IP stack. To have the TCPIP_MAC_Open() succeed after the MAC driver has been initialized by the TCP/IP stack, the configuration symbol DRV_ETHMAC_CLIENTS_NUMBER has to be > 1! But the returned handle is the same one as the TCP/IP stack uses.

Access to the MAC driver in this way is allowed mainly for debugging, diagnostic and statistics purposes. It is possible though to transmit packets this way. But it's not possible to be signalled about incoming RX packets while the stack is running because only the TCP/IP dispatcher will be notified by the RX events.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
 TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
 const TCPIP_MAC_OBJECT* pEthMacObject = TCPIP_STACK_MACObjectGet(netH);
 if(pEthMacObject != 0)
 {    // valid MAC object pointer
      DRV_HANDLE hMac = (*pEthMacObject->TCPIP_MAC_Open)(pEthMacObject->macId, DRV_IO_INTENT_READWRITE);
      if(hMac != DRV_HANDLE_INVALID)
      {   // can use the MAC handle to access a MAC function
          TCPIP_MAC_RX_STATISTICS rxStatistics;
          TCPIP_MAC_TX_STATISTICS txStatistics;
          TCPIP_MAC_RES macRes = (*pEthMacObject->TCPIP_MAC_StatisticsGet)(hMac, &rxStatistics,
&txStatistics);
      }
 }
```

Parameters

Parameters	Description
netH	Interface handle to get the name of.

Function

const TCPIP_MAC_OBJECT* TCPIP_STACK_MACObjectGet(TCPIP_NET_HANDLE netH);

TCPIP_MODULE_SignalFunctionDeregister Function

Deregisters a signal function for a stack module.

File

tcpip_manager.h

C

```
bool TCPIP_MODULE_SignalFunctionDeregister(TCPIP_MODULE_SIGNAL_HANDLE signalHandle);
```

Returns

- true - if the operation succeeded
- false - if the operation failed (i.e., no such module, invalid handle, etc.)

Description

This function deregisters a previous signal function. The caller will no longer be notified of stack internal signals.

Remarks

See the remarks for TCPIP_MODULE_SignalFunctionRegister.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.
signalHandle - a valid signal handle

Example

```
void appSignalFunc(TCPIP_STACK_MODULE moduleId, TCPIP_MODULE_SIGNAL signal)
{
    // process incoming signal for the incoming module
}

TCPIP_MODULE_SIGNAL_HANDLE signalH = TCPIP_MODULE_SignalFunctionRegister( TCPIP_MODULE_HTTP_SERVER,
appSignalFunc);

// when done with this signal notification
TCPIP_MODULE_SignalFunctionDeregister(signalH);
```

Parameters

Parameters	Description
signalHandle	signal handle obtained from a registration call

Function

bool TCPIP_MODULE_SignalFunctionDeregister(TCPIP_MODULE_SIGNAL_HANDLE signalHandle);

TCPIP_MODULE_SignalFunctionRegister Function

Registers a new signal function for a stack module.

File

tcpip_manager.h

C

```
TCPIP_MODULE_SIGNAL_HANDLE TCPIP_MODULE_SignalFunctionRegister(TCPIP_STACK_MODULE moduleId,
TCPIP_MODULE_SIGNAL_FUNC signalF);
```

Returns

- valid handle - if the operation succeeded,
- 0/invalid handle - if the operation failed (i.e., no such module, signal already set, etc.)

Description

This function registers a new signal function. This function will be called and the user notified when a stack internal signal occurs.

Remarks

There is no support for multiple signal functions now. Each module supports just one signal function. A call to register a new module signal function will fail if a function is already registered. TCPIP_MODULE_SignalFunctionDeregister needs to be called first

By default all stack modules, including the stack manager (TCPIP_MODULE_MANAGER) are initialized with a null signal function. Explicit call is needed for setting a module signal function.

A signal handler can be registered for the stack manager itself (TCPIP_MODULE_MANAGER). This will report RX and TMO signals.

The stack internal signaling mechanism is always active and cannot be disabled. This function is called on top of the normal stack signaling mechanism for a module that has a registered signal function.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
void appSignalFunc(TCPIP_STACK_MODULE moduleId, TCPIP_MODULE_SIGNAL signal)
{
    // process incoming signal for the incoming module
}

TCPIP_MODULE_SIGNAL_HANDLE signalH = TCPIP_MODULE_SignalFunctionRegister( TCPIP_MODULE_HTTP_SERVER,
appSignalFunc);
```

Parameters

Parameters	Description
moduleId	module ID
signalF	signal function to be called when an internal signal occurs

Function

TCPIP_MODULE_SIGNAL_HANDLE TCPIP_MODULE_SignalFunctionRegister(TCPIP_STACK_MODULE moduleId, TCPIP_MODULE_SIGNAL_FUNC signalF);

TCPIP_MODULE_SignalGet Function

Returns the current signals for a TCP/IP module.

File

tcpip_manager.h

C

```
TCPIP_MODULE_SIGNAL TCPIP_MODULE_SignalGet(TCPIP_STACK_MODULE moduleId);
```

Returns

The current value of the module's signals.

Description

This function provides a read only value of the current signals for a stack module.

Remarks

The signals are processed and cleared inside the stack. It is not possible to clear a signal that's pending using this API.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_MODULE_SIGNAL currSIgnal = TCPIP_MODULE_SignalGet(TCPIP_MODULE_HTTP_SERVER);
```

Parameters

Parameters	Description
moduleId	module ID

Function

TCPIP_MODULE_SIGNAL TCPIP_MODULE_SignalGet(TCPIP_STACK_MODULE moduleId);

c) Network Status and Control Functions

TCPIP_STACK_IndexToNet Function

Network interface handle from interface number.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_IndexToNet(int netIx);
```

Returns

Network interface handle.

Description

This function converts an interface number to a network interface handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
netIx	Interface index.

Function

TCPIP_NET_HANDLE TCPIP_STACK_IndexToNet(int netIx)

TCPIP_STACK_NetBIOSName Function

Network interface NetBIOS name.

File

tcpip_manager.h

C

```
const char* TCPIP_STACK_NetBIOSName(TCPIP_NET_HANDLE netH);
```

Returns

pointer to the NetBIOS name of that interface

Description

The function returns the network interface NetBIOS name.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
const char* biosName = TCPIP_STACK_NetBIOSName(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get name of.

Function

const char* TCPIP_STACK_NetBIOSName(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetBiosNameSet Function

Sets network interface NetBIOS name.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetBiosNameSet(TCPIP_NET_HANDLE netH, const char* biosName);
```

Returns

- true - if the NetBIOS name of the interface is set
- false - if no such interface or interface is not enabled

Description

This function sets the network interface NetBIOS name.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetBiosNameSet(netH, myBiosName);
```

Parameters

Parameters	Description
netH	Interface handle to set name of.

Function

bool TCPIP_STACK_NetBiosNameSet(TCPIP_NET_HANDLE netH, const char* biosName);

TCPIP_STACK_NetDefaultGet Function

Default network interface handle.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_NetDefaultGet();
```

Returns

The default network interface.

Description

The function returns the current default network interface in the TCP/IP stack. This is the interface on which packets will be transmitted when the internal routing algorithm cannot detect a match for an outgoing packet.

Remarks

This function is intended for multi-homed hosts, with the TCP/IP stack running multiple interfaces.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Function

TCPIP_NET_HANDLE TCPIP_STACK_NetDefaultGet(void)

TCPIP_STACK_NetDefaultSet Function

Sets the default network interface handle.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetDefaultSet(TCPIP_NET_HANDLE netH);
```

Returns

- true - if success
- false - if failed (the old interface does not change)

Description

The function sets the current default network interface in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle.

Function

bool TCPIP_STACK_NetDefaultSet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetHandleGet Function

Network interface handle from a name.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_NetHandleGet(const char* interface);
```

Returns

Network handle.

Description

This function resolves a network interface name to a handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
 or
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("eth0");
```

Parameters

Parameters	Description
interface	The names specified in TCPIP_NETWORK_CONFIG.

Function

TCPIP_NET_HANDLE TCPIP_STACK_NetHandleGet(const char* interface)

TCPIP_STACK_NetIndexGet Function

Network interface number from interface handle.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NetIndexGet(TCPIP_NET_HANDLE hNet);
```

Returns

Index of this network handle in the stack. -1 if invalid.

Description

This function converts a network interface handle to an interface number.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
hNet	Interface handle.

Function

int TCPIP_STACK_NetIndexGet(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetMask Function

Network interface IPv4 address mask.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetMask(TCPIP_NET_HANDLE netH);
```

Returns

- IPv4 address mask of that interface
- 0 if interface is disabled/non-existent

Description

The function returns the IPv4 address mask of the specified interface. If the interface is enabled then it returns the IP address mask of that interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
uint32_t subMask = TCPIP_STACK_NetMask(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get mask of.

Function

uint32_t TCPIP_STACK_NetMask(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetNameGet Function

Network interface name from a handle.

File

tcpip_manager.h

C

```
const char* TCPIP_STACK_NetNameGet(TCPIP_NET_HANDLE netH);
```

Returns

It returns the name associated to that interface handle. Returns 0 if no such name.

Description

This function returns the name associated with the interface handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_IndexToNet(0);
const char* netName = TCPIP_STACK_NetNameGet(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the name of.

Function

const char* TCPIP_STACK_NetNameGet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NumberOfNetworksGet Function

Number of network interfaces in the stack.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NumberOfNetworksGet();
```

Returns

Number of network interfaces.

Description

This function returns the number of interfaces currently active in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Function

int TCPIP_STACK_NumberOfNetworksGet(void)

TCPIP_STACK_EventsPendingGet Function

Returns the currently pending events.

File

tcpip_manager.h

C

```
TCPIP_EVENT TCPIP_STACK_EventsPendingGet(TCPIP_NET_HANDLE hNet);
```

Returns

The currently TCPIP pending events.

Description

This function returns the currently pending events Multiple events can be ORed together as they accumulate.

Remarks

This is the preferred method to get the current pending stack events.

Even with a notification handler in place it's better to use this function to get the current pending events

The returned value is just a momentary value. The pending events can change any time.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_EVENT currEvents = TCPIP_STACK_EventsPendingGet( hNet);
```

Parameters

Parameters	Description
hNet	network handle

Function

TCPIP_EVENT TCPIP_STACK_EventsPendingGet(TCPIP_NET_HANDLE hNet)

TCPIP_STACK_NetIsReady Function

Gets the network interface configuration status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsReady(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists, the corresponding MAC is linked and the interface is properly configured and ready for transfers
- false - if the interface does not exist or interface not ready

Description

This function returns the network interface configuration status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsReady(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetMACRegisterStatisticsGet Function

Get the MAC statistics register data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetMACRegisterStatisticsGet(TCPIP_NET_HANDLE netH, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- true - if the call succeeded
- false - if the call failed (the corresponding MAC does not implement hardware statistics counters).

Description

This function returns the hardware statistics register data of the MAC that is attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_MAC_REG_STATISTICS regStatistics;
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
if(TCPIP_STACK_NetMACRegisterStatisticsGet(netH, &regStatistics))
```

```
{
    // display the hardware statistics registers for the internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use
pRegStatistics	pointer to a pRegEntries that will receive the current hardware statistics registers values. Can be 0, if only the number of supported hardware registers is requested
nEntries	provides the number of TCPIP_MAC_STATISTICS_REG_ENTRY structures present in pRegEntries Can be 0, if only the number of supported hardware registers is requested The register entries structures will be filled by the driver, up to the supported hardware registers.
pHwEntries	pointer to an address to store the number of the statistics registers that the hardware supports It is updated by the driver. Can be 0 if not needed

Function

bool TCPIP_STACK_NetMACRegisterStatisticsGet(TCPIP_NET_HANDLE netH,
TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

TCPIP_STACK_NetAliasNameGet Function

Network interface alias name from a handle.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NetAliasNameGet(TCPIP_NET_HANDLE netH, char* nameBuffer, int buffSize);
```

Returns

It returns the number of characters of the interface alias name. Returns 0 if no such interface.

Description

This function returns the alias name associated with the interface handle.

Remarks

The aliases names are:

- "eth0", "eth1", etc. for Ethernet interfaces
- "wlan0", "wlan1", etc. for Wi-Fi interfaces

See the TCPIP_STACK_IF_NAME_ALIAS_ETH, TCPIP_STACK_IF_NAME_ALIAS_WLAN in tcpip.h for the aliases names.

Alias interface names are at most 8 characters long.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_IndexToNet(0);
char ifName[8];
TCPIP_STACK_NetAliasNameGet(netH, ifName, sizeof(ifName));
```

Parameters

Parameters	Description
netH	Interface handle name to obtain
nameBuffer	buffer to receive the name; could be 0 if only name size is needed
buffSize	size of the provided buffer

Function

int TCPIP_STACK_NetAliasNameGet(TCPIP_NET_HANDLE netH,
char* nameBuffer, int buffSize)

TCPIP_STACK_InitializeDataGet Function

Get the TCP/IP stack initialization data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_InitializeDataGet(SYS_MODULE_OBJ object, TCPIP_STACK_INIT* pStackInit);
```

Returns

- true - if the stack is up and running and the stack init data has been copied
- false - if the stack is down or an error has occurred

Description

This function returns the data that was used for the last call to the TCPIP_STACK_Initialize().

Remarks

The stack does not make a persistent copy of the TCPIP_NETWORK_CONFIG and TCPIP_STACK_MODULE_CONFIG data that are passed in the TCPIP_STACK_INIT at the moment of the stack initialization! It is up to the application to insure that the initialization data is still available after the TCPIP_STACK_Initialize() has been called if this API is used.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_STACK_INIT initData;
TCPIP_STACK_InitializeDataGet(tcpipObject, &initData);
```

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize call
pStackInit	Pointer to an address to store the stack initialization data

Function

bool TCPIP_STACK_InitializeDataGet(SYS_MODULE_OBJ object,
TCPIP_STACK_INIT* pStackInit)

d) Network Up/Down/Linked Functions

TCPIP_STACK_NetIsLinked Function

Gets the network interface link status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsLinked(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists and the corresponding MAC is linked
- false - if the interface does not exist or the link is down

Description

This function returns the network interface link status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsLinked(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetIsUp Function

Gets the network interface up or down status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsUp(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists and is enabled
- false - if the interface does not exist or is disabled

Description

This function returns the network interface up or down (enabled) status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsUp(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetUp Function

Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetUp(TCPIP_NET_HANDLE netH, const TCPIP_NETWORK_CONFIG* pUsrConfig);
```

Returns

- true - if success
- false - if no such network or an error occurred

Description

This function brings a network interface up and perform its initialization.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
netH	Interface handle.
pUsrConfig	pointer to a TCPIP_NETWORK_CONFIG for the interface initialization

Function

bool TCPIP_STACK_NetUp(TCPIP_NET_HANDLE netH, const TCPIP_NETWORK_CONFIG* pUsrConfig)

TCPIP_STACK_NetDown Function

Turns down a network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetDown(TCPIP_NET_HANDLE netH);
```

Returns

- true - if success
- false - if no such network interface or the interface is already down

Description

This function performs the deinitialization of a net interface. As part of this process, the corresponding MAC driver is deinitialized.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle.

Function

bool TCPIP_STACK_NetDown(TCPIP_NET_HANDLE netH)

e) Network Address Status and Control Functions

TCPIP_STACK_NetAddress Function

Network interface IPv4 address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH);
```

Returns

If interface is enabled, it returns the IP address of that interface; otherwise 0 is returned.

Description

If interface is enabled then the function will return the IPv4 address of the network interface.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
uint32_t ipAdd = TCPIP_STACK_NetAddress(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressBcast Function

Network interface broadcast address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressBcast(TCPIP_NET_HANDLE netH);
```

Returns

- Broadcast IP address of that interface
- 0 if no such interface

Description

The function returns the network interface broadcast address. The interface should be enabled for this function to work.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddressBcast(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsPrimary Function

Network interface DNS address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressDnsPrimary(TCPIP_NET_HANDLE netH);
```

Returns

- IPv4 address of the primary DNS server
- 0 if no such interface or interface is down

Description

If interface is enabled then the function will return the IPv4 address of the primary DNS of the network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
myIPAddress = TCPIP_STACK_NetAddressDnsPrimary(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Function

uint32_t TCPIP_STACK_NetAddressDnsPrimary(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsPrimarySet Function

Sets network interface IPv4 DNS address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressDnsPrimarySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface primary IPv4 DNS address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The

network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressDnsPrimarySet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the DNS address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressDnsPrimarySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressDnsSecond Function

Network interface secondary DNS address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressDnsSecond(TCPIP_NET_HANDLE netH);
```

Returns

- The secondary DNS address if success.
- 0 if no such interface or interface is down

Description

If interface is enabled then the function will return the IPv4 address of the secondary DNS of the network interface.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
myIPAddress = TCPIP_STACK_NetAddressDnsSecond(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Function

uint32_t TCPIP_STACK_NetAddressDnsSecond(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsSecondSet Function

Sets network interface IPv4 secondary DNS address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressDnsSecondSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface secondary IPv4 DNS address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressDnsSecondSet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the secondary DNS address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressDnsSecondSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressGateway Function

Network interface IPv4 gateway address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH);
```

Returns

If interface is enabled then it returns the gateway address of that interface. Otherwise, 0 is returned.

Description

If interface is enabled then the function will return the IPv4 gateway address of the network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
uint32_t ipAdd = TCPIP_STACK_NetAddressGateway(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressGatewaySet Function

Sets network interface IPv4 gateway address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressGatewaySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface IPv4 gateway address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressGatewaySet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the gateway address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressGatewaySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressMac Function

Network interface MAC address.

File

tcpip_manager.h

C

```
const uint8_t* TCPIP_STACK_NetAddressMac(TCPIP_NET_HANDLE netH);
```

Returns

- Constant pointer to the MAC address
- 0 if no such interface

Description

The function returns the network interface physical (MAC) address. The interface should be enabled for this function to work.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
const TCPIP_MAC_ADDR* pAdd = TCPIP_STACK_NetAddressMac(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the address of.

Function

const uint8_t* TCPIP_STACK_NetAddressMac(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressMacSet Function

Sets network interface MAC address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressMacSet(TCPIP_NET_HANDLE netH, const TCPIP_MAC_ADDR* pAddr);
```

Returns

- true - if the MAC address of the interface is set
- false - if no such interface or interface is not enabled

Description

This function sets the network interface physical MAC address.

Remarks

One should use extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

This function updates the MAC address in the stack data structures. It does not re-program the MAC with the new address. The MAC programming is done only at MAC initialization for now.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressMacSet(netH, &myMacAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the address of.
pAddr	pointer to a valid physical (MAC) address.

Function

bool TCPIP_STACK_NetAddressMacSet(TCPIP_NET_HANDLE netH, const TCPIP_MAC_ADDR* pAddr)

TCPIP_STACK_NetAddressSet Function

Sets network interface IPv4 address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress, IPV4_ADDR* mask, bool
setDefault);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the associated network IP address and/or mask. If you're changing the network then it's preferred that you set both these values simultaneously to avoid having the stack running with a mismatch between its IP address and mask.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressSet(netH, &myIPAddress, &myIPMask, false);
```

Parameters

Parameters	Description
netH	Interface handle to set address of.
ipAddress	IP address to set (could be NULL to set only the mask)
mask	corresponding network mask to set (could be NULL to set only the IP address)
setDefault	if true, the interface default address/mask is also set

Function

bool TCPIP_STACK_NetAddressSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress, IPV4_ADDR* mask, bool setDefault)

TCPIP_STACK_NetIPv6AddressGet Function

Gets network interface IPv6 address.

File

tcpip_manager.h

C

```
IPV6_ADDR_HANDLE TCPIP_STACK_NetIPv6AddressGet(TCPIP_NET_HANDLE netH, IPV6_ADDR_TYPE addType,
IPV6_ADDR_STRUCT* pAddStruct, IPV6_ADDR_HANDLE addHandle);
```

Returns

- Non-NULL IPV6_ADDR_HANDLE - if an valid IPv6 address was found and the pAddStruct structure was filled with data
- 0 - if no other IPv6 exists or if the supplied IPV6_ADDR_HANDLE is invalid

Description

This function allows the listing of the IPv6 addresses associated with an interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
IPV6_ADDR_STRUCT currAddr;
IPV6_ADDR_HANDLE currHandle;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
char    ipv6AddBuff[44];

currHandle = 0;
do
{
     currHandle = TCPIP_STACK_NetIPv6AddressGet(netH, IPV6_ADDR_TYPE_UNICAST, &currAddr, currHandle);
     if(currHandle)
     {   // have a valid address; display it
         TCPIP_HELPER_IPv6AddressToString(&currAddr.address, ipv6AddBuff, sizeof(ipv6AddBuff));
     }
}while(currHandle != 0);
```

Parameters

Parameters	Description
netH	handle of the interface to retrieve the addresses for
addType	type of address to request IPV6_ADDR_TYPE_UNICAST and IPV6_ADDR_TYPE_MULTICAST supported for now
pAddStruct	structure provided by the user that will be filled with corresponding IPV6_ADDR_STRUCT data
addHandle	an address handle that allows iteration across multiple IPv6 addresses.
On the first call	has to be 0; it will begin the listing of the IPv6 addresses
On subsequent calls	has to be a handle previously returned by a call to this function

Function

IPV6_ADDR_HANDLE TCPIP_STACK_NetIPv6AddressGet(TCPIP_NET_HANDLE netH,
IPV6_ADDR_TYPE addType, IPV6_ADDR_STRUCT* pAddStruct, IPV6_ADDR_HANDLE addHandle);

TCPIP_STACK_ModuleConfigGet Function

Get stack module configuration data.

File

tcpip_manager.h

C

```
size_t TCPIP_STACK_ModuleConfigGet(TCPIP_STACK_MODULE modId, void* configBuff, size_t buffSize, size_t*
pNeededSize);
```

Returns

The number of bytes copied to the user buffer:

- -1 - if the module ID is invalid
- 0 - if the configBuff is NULL or buffSize is less than required
- > 0 - if the call succeeded and the configuration was copied

Description

This function returns the current configuration data of the stack module specified by the corresponding module ID.

Remarks

Currently, only the MAC modules implement this function.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t configBuffer[200];
size_t configSize;
size_t copiedSize;
copiedSize = TCPIP_STACK_ModuleConfigGet(TCPIP_MODULE_MAC_MRF24W, configBuffer,
                                         sizeof(configBuffer), &configSize);
```

Parameters

Parameters	Description
modId	the ID that identifies the requested module
configBuff	pointer to a buffer that will receive the configuration data If this pointer is 0, just the pNeededSize will be updated
buffSize	size of the provided buffer
pNeededSize	pointer to an address to store the number of bytes needed to store this module configuration data. Can be NULL if not needed.

Function

size_t TCPIP_STACK_ModuleConfigGet(TCPIP_STACK_MODULE modId, void* configBuff,

size_t buffSize, size_t* pNeededSize)

TCPIP_STACK_NetMACIdGet Function

Get the MAC ID of the network interface.

File

tcpip_manager.h

C

```
TCPIP_STACK_MODULE TCPIP_STACK_NetMACIdGet(TCPIP_NET_HANDLE netH);
```

Returns

A TCPIP_STACK_MODULE ID that belongs to the MAC of that network interface.

Description

This function returns the module ID of the MAC that's attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_MODULE modId = TCPIP_STACK_NetMACIdGet(netH);
if(modId == TCPIP_MODULE_MAC_PIC32INT)
{
     // an internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use

Function

TCPIP_STACK_MODULE TCPIP_STACK_NetMACIdGet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetMACStatisticsGet Function

Get the MAC statistics data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetMACStatisticsGet(TCPIP_NET_HANDLE netH, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- true - if the call succeeded
- false - if the call failed (the corresponding MAC does not implement statistics counters)

Description

This function returns the statistics data of the MAC that's attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_MAC_RX_STATISTICS rxStatistics;
TCPIP_MAC_TX_STATISTICS txStatistics;
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
if(TCPIP_STACK_NetMACStatisticsGet(netH, &rxStatistics, &txStatistics))
{
     // display the statistics for the internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use
pRxStatistics	pointer to a TCPIP_MAC_RX_STATISTICS that will receive the current RX statistics counters. Can be NULL if not needed.
pTxStatistics	pointer to a TCPIP_MAC_TX_STATISTICS that will receive the current TX statistics counters. Can be NULL if not needed.

Function

bool TCPIP_STACK_NetMACStatisticsGet(TCPIP_NET_HANDLE netH,
TCPIP_MAC_RX_STATISTICS* pRxStatistics, TCPIP_MAC_TX_STATISTICS* pTxStatistics)

TCPIP_STACK_Status Function

Provides the current status of the TCPIP stack module.

File

tcpip_manager.h

C

```
SYS_STATUS TCPIP_STACK_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous initialization operation for the stack has completed
- SYS_STATUS_BUSY - Indicates that a previous initialization operation for the stack has not yet completed

- SYS_STATUS_ERROR - Indicates that the initialization operation has failed and the stack is in an error state

Description

This function provides the current status of the TCPIP stack module.

Remarks

After calling TCPIP_STACK_Initialize, the stack will contiue its initialization process in the TCPIP_STACK_Task routine. This that may take some time. The stack is ready to use only when the TCPIP_STACK_Status returned SYS_STATUS_READY.

Preconditions

The TCPIP_STACK_Initialize function must have been called before calling this function.

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize

Function

SYS_STATUS TCPIP_STACK_Status (SYS_MODULE_OBJ object)

f) Network Structure Storage Functions

TCPIP_STACK_NetConfigGet Function

Get stack network interface configuration data.

File

tcpip_manager.h

C

```
size_t TCPIP_STACK_NetConfigGet(TCPIP_NET_HANDLE netH, void* configStoreBuff, size_t configStoreSize,
size_t* pNeededSize);
```

Returns

- -1 - if the interface is invalid or the stack is not initialized
- 0 - if no data is copied (no supplied buffer of buffer too small)
- > 0 - for success, indicating the amount of data copied.

Description

This function dumps the current configuration data of the network interface specified by the corresponding network handle into the supplied buffer.

Remarks

The function is a helper for retrieving the network configuration data. Its companion function, TCPIP_STACK_NetConfigSet, restores the TCPIP_NETWORK_CONFIG from the dump buffer.

Currently, the data is saved in plain binary format into the supplied buffer. However, the application must not make use of this assumption as it may change in a future release (some compression scheme may be implemented).

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t currConfig[100];
size_t neededSize, result;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
result = TCPIP_STACK_NetConfigGet(hNet, currConfig, sizeof(currConfig), &neededSize);
if(result > 0)
{
     // store the currConfig to some external storage
}
```

Parameters

Parameters	Description
netH	the handle that identifies the requested interface
configStoreBuff	pointer to a buffer that will receive the current configuration data. All the data that's needed to restore a TCPIP_NETWORK_CONFIG structure is stored in this buffer. Can be NULL if only the storage size is needed.
configStoreSize	size of the supplied buffer
pNeededSize	pointer to store the size needed for storage; Can be NULL if not needed

Function

size_t TCPIP_STACK_NetConfigGet(TCPIP_NET_HANDLE netH, void* configStoreBuff,

size_t configStoreSize, size_t* pNeededSize)

TCPIP_STACK_NetConfigSet Function

Restores stack network interface configuration data.

File

tcpip_manager.h

C

```
TCPIP_NETWORK_CONFIG* TCPIP_STACK_NetConfigSet(void* configStoreBuff, void* netConfigBuff, size_t buffSize,
size_t* pNeededSize);
```

Returns

- valid TCPIP_NETWORK_CONFIG pointer (netConfigBuff) if netConfigBuff was successfully updated
- 0 if the netConfigBuff is not supplied or is not large enough

Description

This function restores data from a previously dump buffer and updates the supplied interface configuration. All the data is recovered and constructed into the netConfigBuff (supposing this buffer is large enough). If this operation succeeded, the netConfigBuff can be safely cast to a (TCPIP_NETWORK_CONFIG*).

The structure of the netConfigBuff is as follows:

- A TCPIP_NETWORK_CONFIG structure is created at the very beginning of the buffer.
- All of the necessary fields that are part of the TCPIP_NETWORK_CONFIG will be placed in the buffer itself.

Remarks

The function is a helper for being able to restore the configuration data. Its companion function, TCPIP_STACK_NetConfigGet, saves the TCPIP_NETWORK_CONFIG to a dump buffer.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t currConfig[100];
uint8_t restoreBuff[100];
size_t neededSize, result;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
result = TCPIP_STACK_NetConfigGet(hNet, currConfig, sizeof(currConfig), &neededSize);
if(result > 0)
{
     // store the currConfig buffer to some external storage (neededSize bytes needed)

     // later on restore the configuration
     TCPIP_NETWORK_CONFIG* netConfig;
     // extract the network configuration from the previously saved buffer
     netConfig = TCPIP_STACK_NetConfigSet(currConfig, restoreBuff, sizeof(restoreBuff), neededSize);
     if(netConfig)
     {
         // use this netConfig to initialize a network interface
         TCPIP_STACK_NetUp(hNet, netConfig);
```

```
        }
    }
```

Parameters

Parameters	Description
configStoreBuff	pointer to a buffer that received configuration data from a TCPIP_STACK_NetConfigGet call.
netConfigBuff	pointer to a buffer that will receive the TCPIP_NETWORK_CONFIG data
buffSize	size of the supplied netConfigBuff buffer
pNeededSize	pointer to store the size needed for storage; can be NULL if not needed. If supplied, the pNeededSize will be updated with the actual size that's needed for the netConfigBuff.

Function

TCPIP_NETWORK_CONFIG* TCPIP_STACK_NetConfigSet(void* configStoreBuff, void* netConfigBuff, size_t buffSize, size_t* pNeededSize);

g) Data Types and Constants

TCPIP_LOCAL_MASK_TYPE Enumeration

Mask operation for local/non-local network interface.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_LOCAL_MASK_ZERO,
  TCPIP_LOCAL_MASK_ONE,
  TCPIP_LOCAL_MASK_NET,
  TCPIP_LOCAL_MASK_SET
} TCPIP_LOCAL_MASK_TYPE;
```

Members

Members	Description
TCPIP_LOCAL_MASK_ZERO	use an all zero network mask
TCPIP_LOCAL_MASK_ONE	use an all ones network mask
TCPIP_LOCAL_MASK_NET	use the current network mask
TCPIP_LOCAL_MASK_SET	use the set value for the network mask

Description

Enumeration: TCPIP_LOCAL_MASK_TYPE

These are the available operations that set the local/non-local detection network mask.

TCPIP_NET_HANDLE Type

Defines a network interface handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_NET_HANDLE;
```

Description

Type: TCPIP_NET_HANDLE

Definition of the network handle which clients use to get access to control the interfaces.

TCPIP_EVENT Enumeration

Defines the possible TCPIP event types.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_EV_NONE = 0x0000,
  TCPIP_EV_RX_PKTPEND = 0x0001,
  TCPIP_EV_RX_OVFLOW = 0x0002,
  TCPIP_EV_RX_BUFNA = 0x0004,
  TCPIP_EV_RX_ACT = 0x0008,
  TCPIP_EV_RX_DONE = 0x0010,
  TCPIP_EV_RX_FWMARK = 0x0020,
  TCPIP_EV_RX_EWMARK = 0x0040,
  TCPIP_EV_RX_BUSERR = 0x0080,
  TCPIP_EV_TX_DONE = 0x0100,
  TCPIP_EV_TX_ABORT = 0x0200,
  TCPIP_EV_TX_BUSERR = 0x0400,
  TCPIP_EV_CONN_ESTABLISHED = 0x0800,
  TCPIP_EV_CONN_LOST = 0x1000,
  TCPIP_EV_RX_ALL = (TCPIP_EV_RX_PKTPEND|TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_ACT|
TCPIP_EV_RX_DONE|TCPIP_EV_RX_FWMARK|TCPIP_EV_RX_EWMARK|TCPIP_EV_RX_BUSERR),
  TCPIP_EV_TX_ALL = (TCPIP_EV_TX_DONE|TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR),
  TCPIP_EV_RXTX_ERRORS = (TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_BUSERR|
TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR),
  TCPIP_EV_CONN_ALL = (TCPIP_EV_CONN_ESTABLISHED|TCPIP_EV_CONN_LOST)
} TCPIP_EVENT;
```

Members

Members	Description
TCPIP_EV_NONE = 0x0000	no event
TCPIP_EV_RX_PKTPEND = 0x0001	A receive packet is pending
TCPIP_EV_RX_OVFLOW = 0x0002	RX FIFO overflow (system level latency, no descriptors, etc.)
TCPIP_EV_RX_BUFNA = 0x0004	no RX descriptor available to receive a new packet
TCPIP_EV_RX_ACT = 0x0008	There's RX data available
TCPIP_EV_RX_DONE = 0x0010	A packet was successfully received
TCPIP_EV_RX_FWMARK = 0x0020	the number of received packets is >= than the RX Full Watermark
TCPIP_EV_RX_EWMARK = 0x0040	the number of received packets is <= than the RX Empty Watermark
TCPIP_EV_RX_BUSERR = 0x0080	a bus error encountered during an RX transfer
TCPIP_EV_TX_DONE = 0x0100	A packet was transmitted and it's status is available
TCPIP_EV_TX_ABORT = 0x0200	a TX packet was aborted by the MAC (jumbo/system under-run/excessive defer/late collision/excessive collisions)
TCPIP_EV_TX_BUSERR = 0x0400	a bus error encountered during a TX transfer
TCPIP_EV_CONN_ESTABLISHED = 0x0800	Connection Established
TCPIP_EV_CONN_LOST = 0x1000	Connection Lost
TCPIP_EV_RX_ALL = (TCPIP_EV_RX_PKTPEND|TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_ACT| TCPIP_EV_RX_DONE|TCPIP_EV_RX_FWMARK|TCPIP_EV_RX_EWMARK|TCPIP_EV_RX_BUSERR)	Mask of all RX related events
TCPIP_EV_TX_ALL = (TCPIP_EV_TX_DONE|TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR)	Mask of all TX related events
TCPIP_EV_RXTX_ERRORS = (TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_BUSERR| TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR)	All events showing some abnormal traffic/system condition Action should be taken accordingly by the stack (or the stack user)
TCPIP_EV_CONN_ALL = (TCPIP_EV_CONN_ESTABLISHED|TCPIP_EV_CONN_LOST)	Mask of all Connection related events

Description

TCPIP Stack Events Codes

This enumeration defines all the possible events that can be reported by the TCPIP stack. These are events received by the stack from the network interfaces. They are reported by the MAC driver of the network interface.

Remarks

Depending on the type of the hardware interface, not all events are possible.

Note that specific interfaces can offer specific events and functions to retrieve those events.

TCPIP_EVENT_HANDLE Type

Defines a TCPIP stack event handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_EVENT_HANDLE;
```

Description

Type: TCPIP_EVENT_HANDLE

Definition of an event handle used for event registration by the stack clients.

TCPIP_STACK_EVENT_HANDLER Type

Pointer to a function(handler) that will get called to process an event.

File

tcpip_manager.h

C

```
typedef void (* TCPIP_STACK_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_EVENT, const void* fParam);
```

Returns

None.

Description

TCPIP event notification handler Pointer

Pointer to a function that may be called from within an ISR when a TCP/IP event is available.

Remarks

This function may be invoked from within an ISR. It should be kept as short as possible and it should not include blocking or polling code.

Preconditions

None

Parameters

Parameters	Description
hNet	network handle
tcpEvent	ORed mask of events that occurred
fParam	user passed parameter

Function

void* (TCPIP_NET_HANDLE hNet, TCPIP_EVENT tcpEvent, void* fParam)

TCPIP_Helper_ntohl Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohl(n) TCPIP_Helper_htonl(n)
```

Description

This is macro TCPIP_Helper_ntohl.

TCPIP_Helper_ntohs Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohs(n) TCPIP_Helper_htons(n)
```

Description

This is macro TCPIP_Helper_ntohs.

TCPIP_STACK_IF_NAME_ALIAS_ETH Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_ETH "eth"
```

Description

alias for Ethernet interface

TCPIP_STACK_IF_NAME_ALIAS_UNK Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_UNK "unk"
```

Description

alias for unknown interface

TCPIP_STACK_IF_NAME_ALIAS_WLAN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_WLAN "wlan"
```

Description

alias for Wi-Fi interface

IP_ADDR Type

Backward compatible definition to represent an IPv4 address.

File

tcpip.h

C

```
typedef IPV4_ADDR IP_ADDR;
```

Description

IPv4 Address backward compatible type

This type describes the backward compatible IPv4 address type.

Remarks

None.

IP_ADDRESS_TYPE Enumeration

Definition of the TCP/IP supported address types.

File

tcpip.h

C

```
typedef enum {
  IP_ADDRESS_TYPE_ANY,
  IP_ADDRESS_TYPE_IPV4,
  IP_ADDRESS_TYPE_IPV6
} IP_ADDRESS_TYPE;
```

Members

Members	Description
IP_ADDRESS_TYPE_ANY	either IPv4 or IPv6, unspecified;
IP_ADDRESS_TYPE_IPV4	IPv4 address type
IP_ADDRESS_TYPE_IPV6	IPv6 address type

Description

TCP/IP IP Address type

This type describes the supported IP address types.

Remarks

None.

IP_MULTI_ADDRESS Union

Definition to represent multiple IP addresses.

File

tcpip.h

C

```
typedef union {
  IPV4_ADDR v4Add;
  IPV6_ADDR v6Add;
} IP_MULTI_ADDRESS;
```

Description

TCP/IP multiple Address type

This type describes the TCP/IP multiple address type.

Remarks

None.

IPV4_ADDR Union

Definition to represent an IPv4 address

File

tcpip.h

C

```
typedef union {
  uint32_t Val;
  uint16_t w[2];
  uint8_t v[4];
} IPV4_ADDR;
```

Description

IPv4 Address type

This type describes the IPv4 address type.

Remarks

None.

IPV6_ADDR Union

Definition to represent an IPv6 address.

File

tcpip.h

C

```
typedef union {
  uint8_t v[16];
  uint16_t w[8];
  uint32_t d[4];
} IPV6_ADDR;
```

Description

IPv6 Address type

This type describes the IPv6 address type.

Remarks

None.

IPV6_ADDR_HANDLE Type

Definition to represent an IPv6 address.

File

tcpip.h

C

```
typedef const void* IPV6_ADDR_HANDLE;
```

Description

IPv6 Address handle

This type describes an IPv6 address handle.

Remarks

None.

IPV6_ADDR_SCOPE Enumeration

Definition to represent the scope of an IPv6 address.

File

tcpip.h

C

```
typedef enum {
  IPV6_ADDR_SCOPE_UNKNOWN,
  IPV6_ADDR_SCOPE_INTERFACE_LOCAL,
  IPV6_ADDR_SCOPE_LINK_LOCAL,
  IPV6_ADDR_SCOPE_ADMIN_LOCAL,
  IPV6_ADDR_SCOPE_SITE_LOCAL,
  IPV6_ADDR_SCOPE_ORG_LOCAL,
  IPV6_ADDR_SCOPE_GLOBAL
} IPV6_ADDR_SCOPE;
```

Description

IPv6 Address scope

This type describes the IPv6 address scope.

Remarks

None.

IPV6_ADDR_STRUCT Structure

COmplex type to represent an IPv6 address.

File

tcpip.h

C

```
typedef struct _IPV6_ADDR_STRUCT {
  struct _IPV6_ADDR_STRUCT * next;
  struct _IPV6_ADDR_STRUCT * prev;
  IPV6_ADDR address;
  unsigned long validLifetime;
  unsigned long preferredLifetime;
  unsigned long lastTickTime;
  unsigned char prefixLen;
  struct {
    unsigned char precedence;
    unsigned scope : 4;
    unsigned label : 4;
    unsigned type : 2;
    unsigned temporary : 1;
  } flags;
} IPV6_ADDR_STRUCT;
```

Members

Members	Description
unsigned char precedence;	Allow preferences
unsigned scope : 4;	Link-local, site-local, global.
unsigned label : 4;	Policy label
unsigned type : 2;	Uni-, Any-, Multi-cast
unsigned temporary : 1;	Indicates that the address is temporary (not public)

Description

IPv6 Address Structure

This type defines all the fields of an IPv6 address.

Remarks

None.

IPV6_ADDR_TYPE Enumeration

Definition to represent the type of an IPv6 address.

File

tcpip.h

C

```
typedef enum {
  IPV6_ADDR_TYPE_UNKNOWN,
  IPV6_ADDR_TYPE_UNICAST,
  IPV6_ADDR_TYPE_ANYCAST,
  IPV6_ADDR_TYPE_MULTICAST,
  IPV6_ADDR_TYPE_SOLICITED_NODE_MULTICAST,
  IPV6_ADDR_TYPE_UNICAST_TENTATIVE
} IPV6_ADDR_TYPE;
```

Members

Members	Description
IPV6_ADDR_TYPE_UNKNOWN	Invalid/unknown address type
IPV6_ADDR_TYPE_UNICAST	Only link-local and global are currently valid for unicast

Description

IPv6 Address type

This type describes the possible type of an IPv6 address.

Remarks

None.

TCPIP_NETWORK_CONFIG Structure

Defines the data required to initialize the network configuration.

File

tcpip.h

C

```
typedef struct {
  char* interface;
  char* hostName;
  char* macAddr;
  char* ipAddr;
  char* ipMask;
  char* gateway;
  char* priDNS;
  char* secondDNS;
  char* powerMode;
  TCPIP_NETWORK_CONFIG_FLAGS startFlags;
  const struct TCPIP_MAC_OBJECT_TYPE* pMacObject;
  char* ipv6Addr;
  int ipv6PrefixLen;
  char* ipv6Gateway;
} TCPIP_NETWORK_CONFIG;
```

Members

Members	Description
char* interface;	Pointer to the interface name; could be NULL.
char* hostName;	Valid Host name for this interface to use. Ex: "MCHPBOARD
char* macAddr;	MAC address to use for this interface. Use "00:04:a3:00:00:00" or 0 for the factory preprogrammed address
char* ipAddr;	Static IP address to use. Ex: "169.254.1.1
char* ipMask;	Netmask to use. Ex: "255.255.0.0
char* gateway;	Static Gateway to use. Ex: "169.254.1.1
char* priDNS;	Primary DNS to use. Ex: "169.254.1.1
char* secondDNS;	Secondary DNS to use. Use "0.0.0.0" for none
char* powerMode;	Power Mode to use. Use TCPIP_STACK_IF_POWER_NONE, TCPIP_STACK_IF_POWER_FULL, TCPIP_STACK_IF_POWER_LOW, or TCPIP_STACK_IF_POWER_DOWN
TCPIP_NETWORK_CONFIG_FLAGS startFlags;	flags for interface start-up

const struct TCPIP_MAC_OBJECT_TYPE* pMacObject;	Non-volatile pointer to the MAC driver object associated with this network interface This is the MAC driver that this interface will use
char* ipv6Addr;	static IPv6 address; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified can be NULL if not needed
int ipv6PrefixLen;	subnet prefix length; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified 0 means default value (64) should probably always be 64 as requested by the RFC
char* ipv6Gateway;	default IPv6 gateway address; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified can be NULL if not needed

Description

Multi-Homed Hosts Network Addressing Configuration Data

This data type defines the data required to initialize the network configuration for a specific interface.

Remarks

None.

TCPIP_NETWORK_CONFIG_FLAGS Enumeration

Definition of network configuration start-up flags.

File

tcpip.h

C

```
typedef enum {
  TCPIP_NETWORK_CONFIG_IP_STATIC,
  TCPIP_NETWORK_CONFIG_DHCP_CLIENT_ON,
  TCPIP_NETWORK_CONFIG_ZCLL_ON,
  TCPIP_NETWORK_CONFIG_DHCP_SERVER_ON,
  TCPIP_NETWORK_CONFIG_DNS_CLIENT_ON,
  TCPIP_NETWORK_CONFIG_DNS_SERVER_ON,
  TCPIP_NETWORK_CONFIG_IPV6_ADDRESS
} TCPIP_NETWORK_CONFIG_FLAGS;
```

Members

Members	Description
TCPIP_NETWORK_CONFIG_IP_STATIC	Start the interface with a static IP address No address service is enabled on this interface (DHCPc, ZCLL or DHCPs)
TCPIP_NETWORK_CONFIG_DHCP_CLIENT_ON	DHCP client enabled on this interface
TCPIP_NETWORK_CONFIG_ZCLL_ON	ZeroConf LinkLocal enabled on this interface
TCPIP_NETWORK_CONFIG_DHCP_SERVER_ON	DHCP server enabled on this interface
TCPIP_NETWORK_CONFIG_DNS_CLIENT_ON	DNS CLIENT enabled on this interface
TCPIP_NETWORK_CONFIG_DNS_SERVER_ON	DNS Server Enabled on this Interface
TCPIP_NETWORK_CONFIG_IPV6_ADDRESS	the network configuration contains an IPv6 static address and subnet prefix length

Description

Network Configuration start-up flags

This enumerated type describes the list of the supported TCP/IP network configuration start-up flags.

Remarks

DHCPc, DHCPs and ZCLL are conflicting and cannot be more than one enabled on the same interface! The order of priorities is: DHCPc, ZCLL, DHCPs in case of conflict

Only one of either DNS server or client can be enabled per interface.

Valid values are 0x0001 - 0x8000; 16 bits are maintained only.

TCPIP_STACK_INIT Structure

Definition to represent the TCP/IP stack initialization/configuration structure.

File

tcpip.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  const TCPIP_NETWORK_CONFIG* pNetConf;
  int nNets;
  const TCPIP_STACK_MODULE_CONFIG* pModConfig;
  int nModules;
} TCPIP_STACK_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	system module initialization
const TCPIP_NETWORK_CONFIG* pNetConf;	pointer to array of network configurations
int nNets;	number of networks in the configuration array
const TCPIP_STACK_MODULE_CONFIG* pModConfig;	pointer to array of module configurations
int nModules;	number of modules in the array

Description

TCP/IP stack initialization/configuration structure

This type describes the TCP/IP stack initialization/configuration data type that's passed to the TCP/IP stack at the initialization time.

Remarks

This data structure will be passed to the stack initialization function, TCPIP_STACK_Initialize.

TCPIP_STACK_MODULE Enumeration

List of the TCP/IP stack supported modules.

File

tcpip.h

C

```
typedef enum {
  TCPIP_MODULE_NONE = 0,
  TCPIP_MODULE_IPV4,
  TCPIP_MODULE_IPV6,
  TCPIP_MODULE_LLDP,
  TCPIP_MODULE_ICMPV6,
  TCPIP_MODULE_NDP,
  TCPIP_MODULE_UDP,
  TCPIP_MODULE_TCP,
  TCPIP_MODULE_IGMP,
  TCPIP_MODULE_DHCP_SERVER,
  TCPIP_MODULE_ANNOUNCE,
  TCPIP_MODULE_DNS_CLIENT,
  TCPIP_MODULE_DNS_SERVER,
  TCPIP_MODULE_ZCLL,
  TCPIP_MODULE_MDNS,
  TCPIP_MODULE_NBNS,
  TCPIP_MODULE_SMTP_CLIENT,
  TCPIP_MODULE_SNTP,
  TCPIP_MODULE_FTP_SERVER,
  TCPIP_MODULE_HTTP_SERVER,
  TCPIP_MODULE_HTTP_NET_SERVER,
  TCPIP_MODULE_TELNET_SERVER,
  TCPIP_MODULE_SNMP_SERVER,
  TCPIP_MODULE_SNMPV3_SERVER,
  TCPIP_MODULE_DYNDNS_CLIENT,
  TCPIP_MODULE_BERKELEY,
  TCPIP_MODULE_REBOOT_SERVER,
  TCPIP_MODULE_COMMAND,
  TCPIP_MODULE_IPERF,
  TCPIP_MODULE_TFTP_CLIENT,
  TCPIP_MODULE_DHCPV6_CLIENT,
  TCPIP_MODULE_SMTPC,
  TCPIP_MODULES_NUMBER,
```

```
  TCPIP_MODULE_MAC_START
} TCPIP_STACK_MODULE;
```

Members

Members	Description
TCPIP_MODULE_NONE = 0	unspecified/unknown module
TCPIP_MODULE_LLDP	LLDP module
TCPIP_MODULE_IGMP	IGMP host module
TCPIP_MODULE_ZCLL	Zero Config Link Local
TCPIP_MODULE_MDNS	Bonjour/mDNS
TCPIP_MODULE_TFTP_CLIENT	TFTP client module
TCPIP_MODULE_DHCPV6_CLIENT	DHCPV6 client
TCPIP_MODULE_SMTPC	SMTP (new) client
TCPIP_MODULES_NUMBER	number of modules in the TCP/IP stack itself starting here is list of supported MAC modules and are defined in the tcpip_mac.h

Description

TCP/IP stack supported modules

The following enumeration lists all the modules supported by the TCP/IP stack.

Remarks

None.

TCPIP_STACK_MODULE_CONFIG Structure

Definition to represent a TCP/IP module initialization/configuration structure.

File

tcpip.h

C

```
typedef struct {
  TCPIP_STACK_MODULE moduleId;
  const void * const configData;
} TCPIP_STACK_MODULE_CONFIG;
```

Description

TCP/IP module initialization/configuration structure

This type describes a TCP/IP module initialization/configuration data type that's passed to a TCP/IP module at the stack initialization time.

Remarks

Each stack module will be configured with a user-defined initialization/configuration structure.

TCPIP_MODULE_SIGNAL Enumeration

Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_MODULE_SIGNAL_NONE = 0x0000,
  TCPIP_MODULE_SIGNAL_RX_PENDING = 0x0001,
  TCPIP_MODULE_SIGNAL_TMO = 0x0002,
  TCPIP_MODULE_SIGNAL_ASYNC = 0x0100,
  TCPIP_MODULE_SIGNAL_MASK_ALL = (TCPIP_MODULE_SIGNAL_RX_PENDING | TCPIP_MODULE_SIGNAL_TMO)
} TCPIP_MODULE_SIGNAL;
```

Members

Members	Description
TCPIP_MODULE_SIGNAL_NONE = 0x0000	no pending signal

TCPIP_MODULE_SIGNAL_RX_PENDING = 0x0001	RX packet pending for processing
TCPIP_MODULE_SIGNAL_TMO = 0x0002	module timeout has occurred for modules that implement a state machine advancing on timer signals
TCPIP_MODULE_SIGNAL_ASYNC = 0x0100	Special signals module asynchronous attention required this is a signal that's requested by modules that need special attention module is required to clear this flag when out of the critical processing
TCPIP_MODULE_SIGNAL_MASK_ALL = (TCPIP_MODULE_SIGNAL_RX_PENDING | TCPIP_MODULE_SIGNAL_TMO)	All processing signals mask Note that TCPIP_MODULE_SIGNAL_ASYNC has to be cleared explicitly

Description

Enumeration: TCPIP_MODULE_SIGNAL

These signals are generated by the stack manager towards the internal stack modules; A stack app could use them to provide wake up conditions for a thread waiting for a signal to occur. This is mainly useful when the stack modules tasks functions are executed at the app level and the app needs a way to identify that a module needs attention.

Remarks

Multiple signals could be ORed.

Only 16-bit values are maintained for this type of signal.

TCPIP_MODULE_SIGNAL_FUNC Type

Pointer to a function(handler) that will get called when a stack internal signal occurred.

File

tcpip_manager.h

C

```
typedef void (* TCPIP_MODULE_SIGNAL_FUNC)(TCPIP_MODULE_SIGNAL_HANDLE sigHandle, TCPIP_STACK_MODULE
moduleId, TCPIP_MODULE_SIGNAL signal);
```

Returns

None.

Description

TCPIP module signal function

Pointer to a function that will be called from within the TCP/IP stack when a signal is delivered to a stack module.

Remarks

This function should be kept as short as possible and it should not include blocking or polling code. It's for setting flags/signaling purposes only.

Preconditions

None.

Parameters

Parameters	Description
sigHandle	signal handle obtained from a registration call
moduleId	module that receives the signal
signal	the occurring signal

Function

void* (TCPIP_MODULE_SIGNAL_HANDLE sigHandle, TCPIP_STACK_MODULE moduleId,
TCPIP_MODULE_SIGNAL signal)

TCPIP_MODULE_SIGNAL_HANDLE Type

Defines a TCPIP stack signal function handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_MODULE_SIGNAL_HANDLE;
```

Description

Type: TCPIP_MODULE_SIGNAL_HANDLE

Definition of an signal function handle used for signal registration by the stack clients.

TCPIP_STACK_IF_NAME_97J60 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_97J60 "97J60"
```

Description

This is macro TCPIP_STACK_IF_NAME_97J60.

TCPIP_STACK_IF_NAME_ENCJ60 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ENCJ60 "ENCJ60"
```

Description

This is macro TCPIP_STACK_IF_NAME_ENCJ60.

TCPIP_STACK_IF_NAME_ENCJ600 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ENCJ600 "ENCJ600"
```

Description

This is macro TCPIP_STACK_IF_NAME_ENCJ600.

TCPIP_STACK_IF_NAME_MRF24W Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_MRF24W "MRF24W"
```

Description

This is macro TCPIP_STACK_IF_NAME_MRF24W.

TCPIP_STACK_IF_NAME_NONE Macro

Defines the TCP/IP stack supported network interfaces.

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_NONE 0
```

Description

TCP/IP stack supported network interfaces

The following enumeration lists the names of all the interfaces supported by the TCP/IP stack and aliases.

Remarks

None.

TCPIP_STACK_IF_NAME_PIC32INT Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_PIC32INT "PIC32INT"
```

Description

This is macro TCPIP_STACK_IF_NAME_PIC32INT.

TCPIP_STACK_IF_POWER_DOWN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_DOWN "down"
```

Description

powered down, not started

TCPIP_STACK_IF_POWER_FULL Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_FULL "full"
```

Description

up and running;

TCPIP_STACK_IF_POWER_LOW Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_LOW "low"
```

Description

low power mode; not supported now

TCPIP_STACK_IF_POWER_NONE Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_NONE 0
```

Description

unsupported, invalid

TCPIP_STACK_VERSION_MAJOR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_MAJOR 7
```

Description

TCP/IP stack version

TCPIP_STACK_VERSION_MINOR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_MINOR 25
```

Description

This is macro TCPIP_STACK_VERSION_MINOR.

TCPIP_STACK_VERSION_PATCH Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_PATCH 0
```

Description

This is macro TCPIP_STACK_VERSION_PATCH.

TCPIP_STACK_VERSION_STR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_STR "7.26"
```

Description

This is macro TCPIP_STACK_VERSION_STR.

TCPIP_Helper_ntohll Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohll(ll) TCPIP_Helper_htonll(ll)
```

Description

This is macro TCPIP_Helper_ntohll.

TCPIP_STACK_IF_NAME_MRF24WN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_MRF24WN "MRF24WN"
```

Description

This is macro TCPIP_STACK_IF_NAME_MRF24WN.

Files

Files

Name	Description
tcpip_manager.h	This is the TCP/IP stack manager.
tcpip_helpers.h	TCP/IP Stack helpers header file.
tcpip.h	TCP/IP API definitions.

Description

This section lists the source and header files used by the library.

tcpip_manager.h

This is the TCP/IP stack manager.

Enumerations

	Name	Description
	TCPIP_EVENT	Defines the possible TCPIP event types.
	TCPIP_LOCAL_MASK_TYPE	Mask operation for local/non-local network interface.
	TCPIP_MODULE_SIGNAL	Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.

Functions

	Name	Description
	TCPIP_MODULE_SignalFunctionDeregister	Deregisters a signal function for a stack module.
	TCPIP_MODULE_SignalFunctionRegister	Registers a new signal function for a stack module.
	TCPIP_MODULE_SignalGet	Returns the current signals for a TCP/IP module.
	TCPIP_STACK_Deinitialize	Stack deinitialization function.
	TCPIP_STACK_EventsPendingGet	Returns the currently pending events.
	TCPIP_STACK_HandlerDeregister	Deregisters an event notification handler.
	TCPIP_STACK_HandlerRegister	Sets a new event notification handler.
	TCPIP_STACK_IndexToNet	Network interface handle from interface number.
	TCPIP_STACK_Initialize	Stack initialization function.
	TCPIP_STACK_InitializeDataGet	Get the TCP/IP stack initialization data.
	TCPIP_STACK_MACObjectGet	Returns the network MAC driver object of this interface.
	TCPIP_STACK_ModuleConfigGet	Get stack module configuration data.
	TCPIP_STACK_NetAddress	Network interface IPv4 address.
	TCPIP_STACK_NetAddressBcast	Network interface broadcast address.
	TCPIP_STACK_NetAddressDnsPrimary	Network interface DNS address.
	TCPIP_STACK_NetAddressDnsPrimarySet	Sets network interface IPv4 DNS address.
	TCPIP_STACK_NetAddressDnsSecond	Network interface secondary DNS address.
	TCPIP_STACK_NetAddressDnsSecondSet	Sets network interface IPv4 secondary DNS address.

	TCPIP_STACK_NetAddressGateway	Network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressGatewaySet	Sets network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressMac	Network interface MAC address.
	TCPIP_STACK_NetAddressMacSet	Sets network interface MAC address.
	TCPIP_STACK_NetAddressSet	Sets network interface IPv4 address.
	TCPIP_STACK_NetAliasNameGet	Network interface alias name from a handle.
	TCPIP_STACK_NetBIOSName	Network interface NetBIOS name.
	TCPIP_STACK_NetBiosNameSet	Sets network interface NetBIOS name.
	TCPIP_STACK_NetConfigGet	Get stack network interface configuration data.
	TCPIP_STACK_NetConfigSet	Restores stack network interface configuration data.
	TCPIP_STACK_NetDefaultGet	Default network interface handle.
	TCPIP_STACK_NetDefaultSet	Sets the default network interface handle.
	TCPIP_STACK_NetDown	Turns down a network interface.
	TCPIP_STACK_NetHandleGet	Network interface handle from a name.
	TCPIP_STACK_NetIndexGet	Network interface number from interface handle.
	TCPIP_STACK_NetIPv6AddressGet	Gets network interface IPv6 address.
	TCPIP_STACK_NetIsLinked	Gets the network interface link status.
	TCPIP_STACK_NetIsReady	Gets the network interface configuration status.
	TCPIP_STACK_NetIsUp	Gets the network interface up or down status.
	TCPIP_STACK_NetMACIdGet	Get the MAC ID of the network interface.
	TCPIP_STACK_NetMACRegisterStatisticsGet	Get the MAC statistics register data.
	TCPIP_STACK_NetMACStatisticsGet	Get the MAC statistics data.
	TCPIP_STACK_NetMask	Network interface IPv4 address mask.
	TCPIP_STACK_NetNameGet	Network interface name from a handle.
	TCPIP_STACK_NetUp	Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.
	TCPIP_STACK_NumberOfNetworksGet	Number of network interfaces in the stack.
	TCPIP_STACK_SetLocalMasks	Sets the local/non-local masks for network interface.
	TCPIP_STACK_SetLocalMasksType	Sets the local/non-local masks type for network interface.
	TCPIP_STACK_Status	Provides the current status of the TCPIP stack module.
	TCPIP_STACK_Task	TCP/IP Stack task function.
	TCPIP_STACK_VersionGet	Gets the TCP/IP stack version in numerical format.
	TCPIP_STACK_VersionStrGet	Gets the TCP/IP stack version in string format.

Types

	Name	Description
	TCPIP_EVENT_HANDLE	Defines a TCPIP stack event handle.
	TCPIP_MODULE_SIGNAL_FUNC	Pointer to a function(handler) that will get called when a stack internal signal occurred.
	TCPIP_MODULE_SIGNAL_HANDLE	Defines a TCPIP stack signal function handle.
	TCPIP_NET_HANDLE	Defines a network interface handle.
	TCPIP_STACK_EVENT_HANDLER	Pointer to a function(handler) that will get called to process an event.

Description

Microchip TCP/IP Stack Definitions

The TCP/IP manager provides access to the stack initialization, configuration and status. It also process the packets received from different network interfaces (MAC drivers) and dispatches them accordingly based on their type to the proper higher layer in the stack.

File Name

tcpip_manager.h

Company

Microchip Technology Inc.

tcpip_helpers.h

TCP/IP Stack helpers header file.

Functions

	Name	Description
	TCPIP_Helper_Base64Decode	Helper function to decode a Base-64 encoded array.
	TCPIP_Helper_Base64Encode	Helper function to encode to Base-64.
	TCPIP_Helper_FormatNetBIOSName	Formats a string to a valid NetBIOS name.
	TCPIP_Helper_htonl	Conversion routines from network order to host order and reverse.
	TCPIP_Helper_htonll	This is function TCPIP_Helper_htonll.
	TCPIP_Helper_htons	This is function TCPIP_Helper_htons.
	TCPIP_Helper_IPAddressToString	Converts an IPV4 address to an ASCII string.
	TCPIP_Helper_IPv6AddressToString	Converts an IPv6 address to a string representation.
	TCPIP_Helper_IsBcastAddress	Checks if an IPv4 address is a broadcast address.
	TCPIP_Helper_IsMcastAddress	Checks if an IPv4 address is a multicast address.
	TCPIP_Helper_IsPrivateAddress	Detects a private (non-routable) address.
	TCPIP_Helper_MACAddressToString	Converts a MAC address to a string.
	TCPIP_Helper_ntohl	!defined(__PIC32MX__)
	TCPIP_Helper_ntohll	This is function TCPIP_Helper_ntohll.
	TCPIP_Helper_ntohs	This is function TCPIP_Helper_ntohs.
	TCPIP_Helper_SecurePortGetByIndex	Returns the secure port belonging to a specified index.
	TCPIP_Helper_SecurePortSet	Sets the required port secure connection status.
	TCPIP_Helper_StringToIPAddress	Converts an ASCII string to an IPV4 address.
	TCPIP_Helper_StringToIPv6Address	Converts a string to an IPv6 address.
	TCPIP_Helper_StringToMACAddress	Converts a string to an MAC address.
	TCPIP_Helper_TCPSecurePortGet	Checks if the required TCP port is a secure port.
	TCPIP_Helper_UDPSecurePortGet	Checks if the required UDP port is a secure port.

Macros

	Name	Description
	TCPIP_Helper_ntohl	This is macro TCPIP_Helper_ntohl.
	TCPIP_Helper_ntohll	This is macro TCPIP_Helper_ntohll.
	TCPIP_Helper_ntohs	This is macro TCPIP_Helper_ntohs.

Description

Header file for tcpip_stack_helpers

This file provides the APIs for the TCP/IP Stack helpers.

File Name

tcpip_helpers.h

Company

Microchip Technology Inc.

tcpip.h

TCP/IP API definitions.

Enumerations

	Name	Description
	IP_ADDRESS_TYPE	Definition of the TCP/IP supported address types.
	IPV6_ADDR_SCOPE	Definition to represent the scope of an IPv6 address.
	IPV6_ADDR_TYPE	Definition to represent the type of an IPv6 address.
	TCPIP_NETWORK_CONFIG_FLAGS	Definition of network configuration start-up flags.
	TCPIP_STACK_MODULE	List of the TCP/IP stack supported modules.

Macros

	Name	Description
	TCPIP_STACK_IF_NAME_97J60	This is macro TCPIP_STACK_IF_NAME_97J60.

	TCPIP_STACK_IF_NAME_ALIAS_ETH	alias for Ethernet interface
	TCPIP_STACK_IF_NAME_ALIAS_UNK	alias for unknown interface
	TCPIP_STACK_IF_NAME_ALIAS_WLAN	alias for Wi-Fi interface
	TCPIP_STACK_IF_NAME_ENCJ60	This is macro TCPIP_STACK_IF_NAME_ENCJ60.
	TCPIP_STACK_IF_NAME_ENCJ600	This is macro TCPIP_STACK_IF_NAME_ENCJ600.
	TCPIP_STACK_IF_NAME_MRF24W	This is macro TCPIP_STACK_IF_NAME_MRF24W.
	TCPIP_STACK_IF_NAME_MRF24WN	This is macro TCPIP_STACK_IF_NAME_MRF24WN.
	TCPIP_STACK_IF_NAME_NONE	Defines the TCP/IP stack supported network interfaces.
	TCPIP_STACK_IF_NAME_PIC32INT	This is macro TCPIP_STACK_IF_NAME_PIC32INT.
	TCPIP_STACK_IF_POWER_DOWN	powered down, not started
	TCPIP_STACK_IF_POWER_FULL	up and running;
	TCPIP_STACK_IF_POWER_LOW	low power mode; not supported now
	TCPIP_STACK_IF_POWER_NONE	unsupported, invalid
	TCPIP_STACK_VERSION_MAJOR	TCP/IP stack version
	TCPIP_STACK_VERSION_MINOR	This is macro TCPIP_STACK_VERSION_MINOR.
	TCPIP_STACK_VERSION_PATCH	This is macro TCPIP_STACK_VERSION_PATCH.
	TCPIP_STACK_VERSION_STR	This is macro TCPIP_STACK_VERSION_STR.

Structures

	Name	Description
	_IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	TCPIP_NETWORK_CONFIG	Defines the data required to initialize the network configuration.
	TCPIP_STACK_INIT	Definition to represent the TCP/IP stack initialization/configuration structure.
	TCPIP_STACK_MODULE_CONFIG	Definition to represent a TCP/IP module initialization/configuration structure.

Types

	Name	Description
	IP_ADDR	Backward compatible definition to represent an IPv4 address.
	IPV6_ADDR_HANDLE	Definition to represent an IPv6 address.

Unions

	Name	Description
	IP_MULTI_ADDRESS	Definition to represent multiple IP addresses.
	IPV4_ADDR	Definition to represent an IPv4 address
	IPV6_ADDR	Definition to represent an IPv6 address.

Description

Microchip TCP/IP Stack Include File

This is the global TCP/IP header file that any user of the TCP/IP API should include. It contains the basic TCP/IP types and data structures and includes all the of the TCP/IP stack modules.

File Name

tcpip.h

Company

Microchip Technology Inc.

NBNS Module

This section describes the TCP/IP Stack Library NBNS module.

Introduction

TCP/IP Stack Library NetBIOS (NBNS) Module for Microchip Microcontrollers

This library provides the API of the NBNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example. in the "TCP/IP Demo App" demonstration project, the development board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

Using the Library

This topic describes the basic architecture of the NBNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `nbns.h`

The interface to the NBNS TCP/IP Stack library is defined in the `nbns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the NBNS TCP/IP Stack library should include `tcpip.h`.

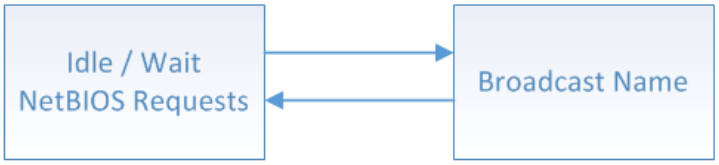
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the NBNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

NBNS 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 NBNS module.

Library Interface Section	Description
Functions	This section provides a routine that performs NBNS module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing This API

Configuring the Library

Macros

	Name	Description
	TCPIP_NBNS_TASK_TICK_RATE	NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the NBNS TCP/IP Stack is based on the `file tcpip_config.h`.

This header file contains the configuration selection for the NBNS TCP/IP Stack. Based on the selections made, the NBNS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the NBNS TCP/IP Stack.

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.

TCPIP_NBNS_TASK_TICK_RATE Macro

File

nbns_config.h

C

```
#define TCPIP_NBNS_TASK_TICK_RATE 110
```

Description

NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the NBNS module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/nbns.c`	NBNS 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 NBNS module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_NBNS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_NBNS_MODULE_CONFIG	Placeholder for NBNS configuration upgrades.

Description

This section describes the Application Programming Interface (API) functions of the NBNS module.

Refer to each section for a detailed description.

a) Functions

TCPIP_NBNS_Task Function

Standard TCP/IP stack module task function.

File

nbns.h

C

```
void TCPIP_NBNS_Task();
```

Returns

None.

Description

This function performs NBNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The NBNS module should have been initialized.

Function

void TCPIP_NBNS_Task(void)

b) Data Types and Constants

TCPIP_NBNS_MODULE_CONFIG Structure

Placeholder for NBNS configuration upgrades.

File

nbns.h

C

```
typedef struct {
} TCPIP_NBNS_MODULE_CONFIG;
```

Description

TCPIP_NBNS_MODULE_CONFIG Structure Typedef

This type definition provides a placeholder for NBNS configuration upgrades.

Remarks

None.

Files

Files

Name	Description
nbns.h	The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet.
nbns_config.h	NBNS configuration file

Description

This section lists the source and header files used by the library.

nbns.h

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet.

Functions

	Name	Description
	TCPIP_NBNS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_NBNS_MODULE_CONFIG	Placeholder for NBNS configuration upgrades.

Description

NetBIOS Name Service (NBNS) Server public API

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example. in the "TCP/IP Demo App" demonstration project, the demo board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

File Name

nbns.h

Company

Microchip Technology Inc.

nbns_config.h

NBNS configuration file

Macros

	Name	Description
	TCPIP_NBNS_TASK_TICK_RATE	NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

NetBIOS Name Service (NBNS) Configuration file

This file contains the NBNS module configuration options

File Name

nbns_config.h

Company

Microchip Technology Inc.

NDP Module

This section describes the TCP/IP Stack Library NDP module.

Introduction

TCP/IP Stack Library NDP Module for Microchip Microcontrollers

This library provides the API of the NDP (Neighbor Discovery Protocol) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This document provides NDP (Neighbor Discovery Protocol) for IP version 6. IPv6 nodes on the same link use neighbor discovery to discover each other's presence. RFC - 4861.

It is responsible for:

- Address Auto configuration of nodes
- Discovery of other nodes in the link (It can be a Router Discovery or Neighbor Discovery)
- Determining the link-layer address of other nodes
- Duplicate address detection (DAD)
- Finding available routers and Domain Name System (DNS) servers
- Address Prefix discovery
- Parameter Discovery (Such as Link MTU or Hop limit)

Comparing with IPv4:

- NDP is a substitute of ARP (Address Resolution protocol). This new mechanism uses a mix of ICMPv6 and multicast addresses to discover the IPv6 node on same link.
- NDP includes Neighbor Unreachability Detection (NUD) , thus improving the robustness of packet delivery.
- Unlike IPv4 broadcast addresses, IPv6 address resolution multicasts are spread over 4 billion (2^32) multicast addresses, greatly reducing address resolution-related interrupts on nodes other than the target. Moreover, non-IPv6 machines should not be interrupted at all.
- Neighbor Discovery defines five different ICMPv6 packet types.

Five Different ICMPv6 Packet Types:

- **Router Solicitation** - Hosts inquire with Router Solicitation message to locate routers on the attached link
- **Router Advertisement** - Router advertise their presence periodically to all the nodes or in response to the Router Solicitation message
- **Neighbor Solicitation** - Neighbor solicitations are used by nodes to determine the Link Layer address of a neighbor, or to verify that a neighbor is still reachable via a cached Link Layer address
- **Neighbor Advertisement** - Neighbor advertisements are used by nodes to respond to a Neighbor Solicitation message
- **Redirect** - Routers may inform hosts of a better first hop router for a destination

Using the Library

This topic describes the basic architecture of the NDP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ndp.h`

The interface to the NDP TCP/IP Stack Library is defined in the `ndp.h` header file. Any C language source (`.c`) file that uses the NDP TCP/IP library should include `ipv6.h`, `icmpv6.h`, and `ndp.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack Library interacts with the framework.

Abstraction Model

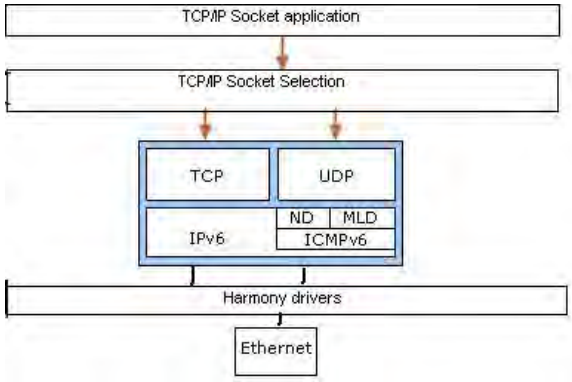
This library provides the API of the NDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

NDP with IPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



Link Layer Neighbor Discovery

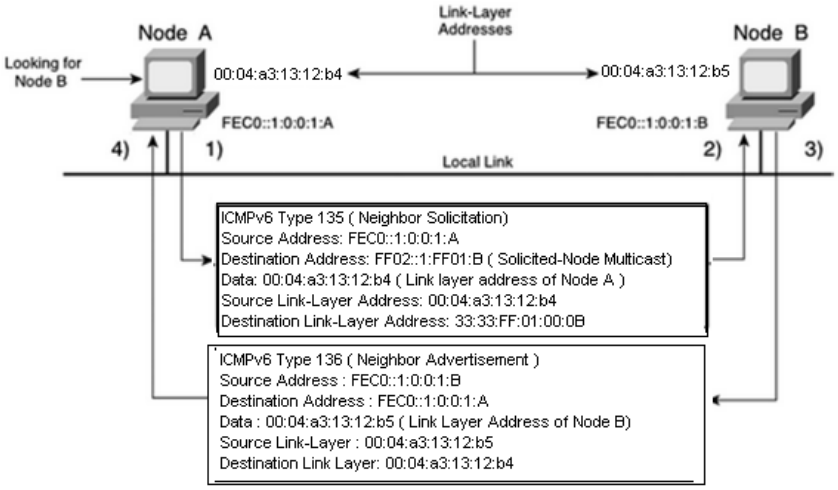
1) Using the address FEC0::1:0:0:1 :A, node A wants to deliver packets to destination node B using the IPv6 address FEC0::1 :0:0:1 :B on the same local link. However, node A does not know node B's link-layer address. Node A sends an ICMPv6 Type

135 message (neighbor solicitation) on the local link using its site-local address FEC0::1:0:0:1:A as the IPv6 source address, the solicited-node multicast address FF02::1 :FF01:B corresponding to the target address FEC0::1 :0:0:1 :B as the destination IPv6 address, and the source link-layer address 00:04:a3:13:12:b4 of the sender, node A, as data of the ICMPv6 message. The source link-layer address of this frame is the link-layer address 00:04:a3:13:12:b4 of node A. The destination link-layer address 33:33:FF:01 :00:0B of this frame uses multicast mapping of the destination IPv6 address FF02::1 :FF01 :B.

2) Node B, which is listening to the local link for multicast addresses, intercepts the neighbor solicitation message because the destination IPv6 address FF02::1:FF01:B represents the solicited-node multicast address corresponding to its IPv6 address FEC0::1:0:0:1:B.

3) Node B replies by sending a neighbor advertisement message using its site-local address FEC0::1 :0:0:1 :B as the IPv6 source address and the site-local address FEC0::1 :0:0:1 :A as the destination IPv6 address. It also includes its link-layer address 00:04:a3:13:12:b5 in the ICMPv6 message. After receiving neighbor solicitation and neighbor advertisement messages, node A and node B know each other's link-layer addresses.

Learned link-layer addresses are kept in a neighbor discovery table (neighbor cache). Therefore, the nodes can communicate on the local link. The neighbor solicitation message is also used by nodes to verify the reachability of neighbor nodes in the neighbor discovery table (neighbor cache). However, the unicast addresses of the neighbor nodes are used as destination IPv6 addresses in ICMPv6 messages instead of solicited-node multicast addresses in this situation. It is possible for a node that changes its link-layer address to inform all other neighbor nodes on the local link by sending a neighbor advertisement message using the all-nodes multicast address FF02::1 . The neighbor discovery table of the nodes on the local link is updated with the new link-layer address.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6, ICMPv6, and NDP module.

Library Interface Section	Description
Reachability Functions	Reachability routine functionality

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME	5 s
	TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME	1 s
	TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY	1 s
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS	3 transmissions
	TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_REACHABLE_TIME	30 s
	TCPIP_IPV6_NDP_RETRANS_TIMER	1 s
	TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL	4 s
	TCPIP_IPV6_NDP_TASK_TIMER_RATE	The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours

Description

The configuration of the TCP/IP Stack NDP module is based on the file `ndp_config.h`.

This header file contains the configuration selection for the TCP/IP Stack IP. Based on the selections made, the TCP/IP Stack IP may support the selected features. These configuration settings will apply to all instances of the TCP/IP Stack IP.

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.

TCPIP_IPV6_MTU_INCREASE_TIMEOUT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_MTU_INCREASE_TIMEOUT 600ul           // 600 seconds
```

Description

600 seconds

TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME 5u              // 5 s
```

Description

5 s

TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME 1u              // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT 3u              // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT 3u              // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY 1u              // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS 3u              // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT 3u              // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_REACHABLE_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_REACHABLE_TIME 30u            // 30 s
```

Description

30 s

TCPIP_IPV6_NDP_RETRANS_TIMER Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_RETRANS_TIMER 1u              // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL 4u              // 4 s
```

Description

4 s

TCPIP_IPV6_NDP_TASK_TIMER_RATE Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_TASK_TIMER_RATE (32)
```

Description

The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS (60 * 60 * 2)
```

Description

Sets the lifetime to 2 hours

Building the Library

This section lists the files that are available in the NDP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/ndp.c`	NDP 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 NDP module depends on the following modules:

- TCP/IP Stack Library
- IPv6 Module

Library Interface

Reachability Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

Description

This section describes the Application Programming Interface (API) functions of the NDP module.

Refer to each section for a detailed description.

Reachability Functions

TCPIP_NDP_NborReachConfirm Function

Confirms that a neighbor is reachable.

File

ndp.h

C

```
void TCPIP_NDP_NborReachConfirm(TCPIP_NET_HANDLE netH, const IPV6_ADDR * address);
```

Returns

None.

Description

This function is used by upper-layer protocols to indicate that round-trip communications were confirmed with a neighboring node.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
pNetIf	The interface the neighbor is on.
address	The address of the neighbor.

Function

void TCPIP_NDP_NborReachConfirm (TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

Files

Files

Name	Description
ndp.h	IPv6 Internet Communication Message Neighbor Discovery Protocol (NDP).
ndp_config.h	NDP configuration file

Description

This section lists the source and header files used by the library.

ndp.h

IPv6 Internet Communication Message Neighbor Discovery Protocol (NDP).

Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

Description

Neighbor Discovery Protocol (NDP) API Header File

Neighbor Discovery Protocol (NDP) in IPv6 is the substitute of as ARP (which is used in IPv4 for address resolve). NDP is used discover link local addresses of the IPv6 nodes present in the local link using a mix of ICMPv6 messages and multicast addresses, stateless auto-configuration and router redirection.

File Name

ndp.h

Company

Microchip Technology Inc.

ndp_config.h

NDP configuration file

Macros

	Name	Description
	TCPIP_IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME	5 s

	TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME	1 s
	TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY	1 s
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS	3 transmissions
	TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_REACHABLE_TIME	30 s
	TCPIP_IPV6_NDP_RETRANS_TIMER	1 s
	TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL	4 s
	TCPIP_IPV6_NDP_TASK_TIMER_RATE	The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours

Description

Neighbor Discovery Protocol (NDP) Configuration file

This file contains the NDP module configuration options

File Name

ndp_config.h

Company

Microchip Technology Inc.

Reboot Module

This section describes the TCP/IP Stack Library Reboot module.

Introduction

TCP/IP Stack Library Reboot Module for Microchip Microcontrollers

This library provides the API of the Reboot module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Reboot module will allow a user to remotely reboot the PIC microcontroller that is running the TCP/IP stack. This feature is primarily used for bootloader applications, which must reset the microcontroller to enter the bootloader code section. This module will execute a task that listens on a specified UDP port for a packet, and then reboots if it receives one.

Using the Library

This topic describes the basic architecture of the Reboot TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_reboot_config.h`

The interface to the Reboot TCP/IP Stack library is defined in the `tcpip_reboot_config.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Reboot TCP/IP Stack library should include `tcpip.h`.

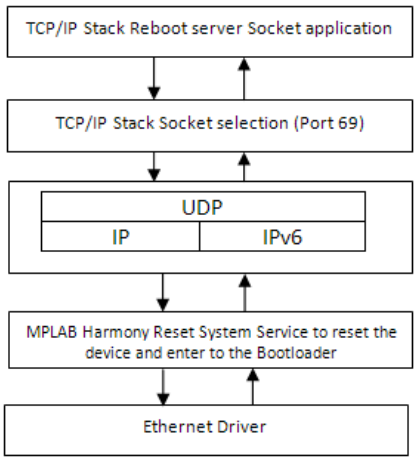
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Reboot TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Reboot Software Abstraction Block Diagram



Configuring the Library

Macros

	Name	Description
	TCPIP_REBOOT_MESSAGE	the mesage needed to be sent accross the net to reboot the machine
	TCPIP_REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

	TCPIP_REBOOT_TASK_TICK_RATE	the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the Reboot TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Reboot TCP/IP Stack Library. Based on the selections made, the Reboot TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Reboot TCP/IP Stack 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.

TCPIP_REBOOT_MESSAGE Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_MESSAGE "MCHP Reboot"
```

Description

the mesage needed to be sent accross the net to reboot the machine

TCPIP_REBOOT_SAME_SUBNET_ONLY Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_SAME_SUBNET_ONLY
```

Description

For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

TCPIP_REBOOT_TASK_TICK_RATE Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_TASK_TICK_RATE 1130
```

Description

the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the Reboot module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/tcpip_reboot.c`	Reboot 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 Reboot module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Files

Files

Name	Description
tcpip_reboot_config.h	Configuration file

Description

This section lists the source and header files used by the library.

tcpip_reboot_config.h

Configuration file

Macros

	Name	Description
	TCPIP_REBOOT_MESSAGE	the mesage needed to be sent accross the net to reboot the machine
	TCPIP_REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.
	TCPIP_REBOOT_TASK_TICK_RATE	the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Reboot Configuration file

This file contains the Reboot module configuration options

File Name

tcpip_reboot_config.h

Company

Microchip Technology Inc.

SMTP Module

This section describes the TCP/IP Stack Library SMTP module.

Introduction

TCP/IP Stack Library Simple Mail Transfer Protocol (SMTP) Module for Microchip Microcontrollers

This library provides the API of the SMTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTP client requires access to a local mail server (such as `mail.yourdomain.com`) for reliable operation. Your MPLAB Harmony or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

Using the Library

This topic describes the basic architecture of the SMTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `smtp.h`

The interface to the SMTP TCP/IP Stack library is defined in the `smtp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SMTP TCP/IP Stack library should include `tcpip.h`.

Library File:

The SMTP TCP/IP Stack library archive (`.a`) file is installed with MPLAB Harmony.

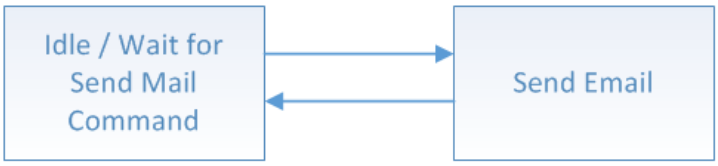
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SMTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SMTP 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 SMTP module.

Library Interface Section	Description
Functions	Routines to configure this module
Data Types and Constants	This section provides various definitions describing this API

SMTP Client Examples

SMTP Client examples.

Description

The following two examples demonstrate the use of the SMTP client in different scenarios. The first, and simpler example, sends a short message whose contents are all located in RAM at once.

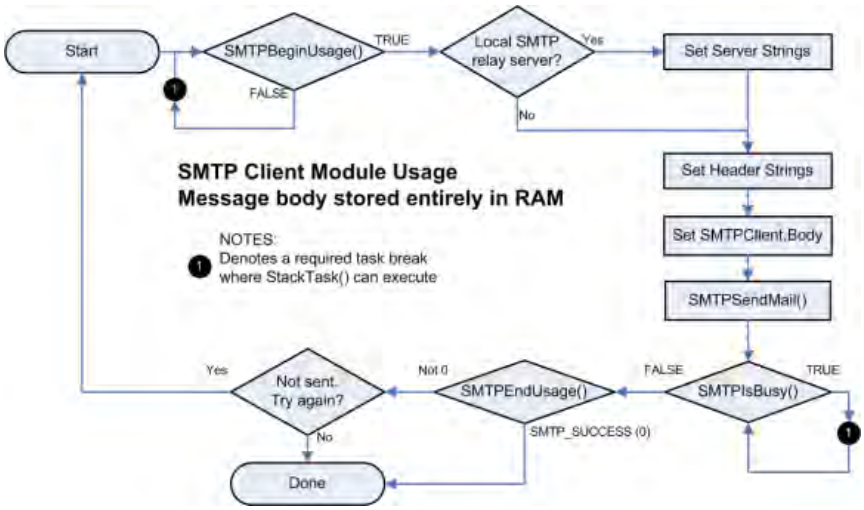
The second example is more involved and demonstrates generating a message on the fly in the case where the entire message cannot fit into RAM at once. The SMTPDemo example provided in `MainDemo.c` sends a brief e-mail message indicating the current status of the board's buttons.

SMTP Client Short Message Example

The SMTP client API is simplified when messages can be buffered entirely in RAM. The SMTPDemo example provided in `MainDemo.c` sends a brief e-mail message indicating the current status of the board's buttons. This document will walk through that example.

Make sure STACK_USE_SMTP_CLIENT is uncommented in `tcpip_config.h` before continuing.

The following diagram provides an overview of the process:



1. First, call SMTPBeginUsage to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.
2. Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials. If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to SPAM prevention measures put in place by most ISPs and network administrators.
3. Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add. Finally, set SMTPClient.Body to the message to be sent.
4. At this point, verify that SMTPClient.ROMPointers is correctly configured for any strings that are stored in program memory. Once the message is ready to send, call SMTPSendMail to instruct the SMTP client to begin transmission.
5. The application must now call SMTPIsBusy until it returns FALSE. Each time TRUE is returned, return to the main loop and wait for StackTask to execute again. This allows the SMTP server to continue its work in a cooperative multitasking manner. Once FALSE is returned, call SMTPEndUsage to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. The Server and To fields must be set in SMTPDemo in order for the message to be properly delivered. Once this is done, holding down BUTTON2 and BUTTON3 simultaneously (the left-most two buttons) will begin sending the message. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

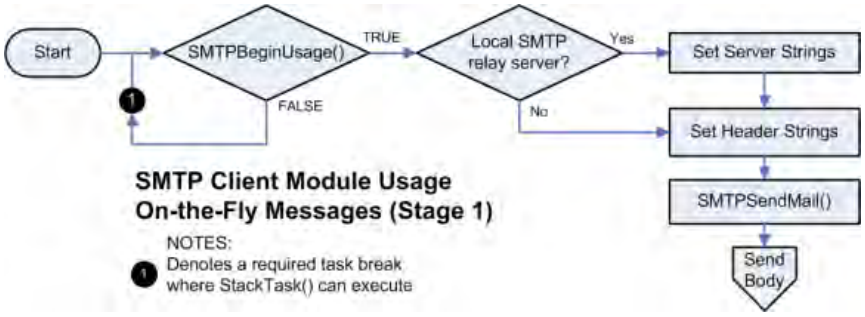
SMTP Client Long Message Example

The SMTP client API is capable of sending messages that do not fit entirely in RAM. To do so, the application must manage its output state and only write as many bytes as are available in the buffer at a time. The second SMTPDemo example provided in `MainDemo.c` sends a message that is a dump of all contents of the PIC microcontroller's RAM. This example is currently commented out. Comment out the previous Short Message Example and uncomment the Long Message Example. This document will walk through sending a longer message.

Make sure STACK_USE_SMTP_CLIENT is uncommented in `tcpip_config.h` before continuing.

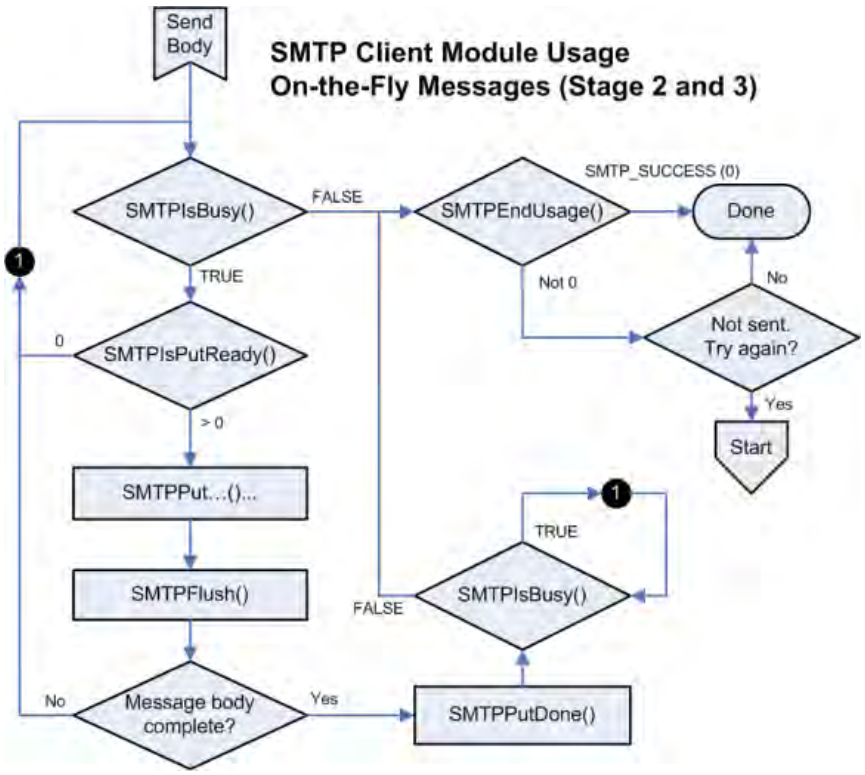
Sending longer messages is divided into three stages. The first stage configures the SMTP client to send the message. The second stage sends the message in small chunks as buffer space is available. The final stage finishes the transmission and determines whether or not the message was successful.

The following diagram illustrates the first stage:



1. The first stage is largely similar to the first few steps in sending a short message. First, call SMTPBeginUsage to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.
2. Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials. If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to SPAM prevention measures put in place by most ISPs and network administrators.
3. Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add.
4. The next portion of the process differs. Ensure that SMTPClient.Body remains set to its default (NULL). At this point, call SMTPSendMail to open a connection to the remote server and transmit the headers. The application is now ready to proceed to the second stage and send the message body.

The following diagram provides an overview of stage two and three:



1. Upon entering stage two, the application should call SMTPIsBusy to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call SMTPIsPutReady to determine how many bytes are available in the TX buffer. If no bytes are available, return to the main loop so that StackTask can transmit the data to the remote node and free up the buffer.
2. If space is available, any combination of the SMTPPut, SMTPPutArray, SMTPPutROMArray, SMTPPutString, and SMTPPutROMString functions may be called to transmit the message. These functions return the number of bytes successfully written. Use this value, along with the value originally returned from SMTPIsPutReady to track how much free space remains in the TX buffer. Once the buffer is depleted, call SMTPFlush to force the data written to be sent.
3. The SMTP client module can accept as much data as the TCP TX FIFO can hold. This is determined by the socket initializer for TCP_PURPOSE_DEFAULT type sockets in `tcpip_config.h`, which defaults to 200 bytes.
4. If the TX buffer is exhausted before the message is complete, return to the main loop so that StackTask may transmit the data to the remote node and free up the buffer. Upon return, go to the beginning of the second stage to transmit the next portion of the message.

Once the message is complete, the application will move to the third stage. Call SMTPPutDone to inform the SMTP client that no more data remains. Then call SMTPIsBusy repeatedly. Each time TRUE is returned, return to the main loop and wait for StackTask to execute again. Once FALSE is returned, the message transmission has completed and the application must call SMTPEndUsage to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. Set the Server and To fields in SMTPDemo, and ensure that these fields are being properly assigned to SMTPClient struct. The demonstration works exactly the same way as the previous one, with BUTTON2 and BUTTON3 held down simultaneously (the left-most two buttons) kicking off the state machine. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will turn ON; otherwise, it will remain OFF.

Configuring the Library

Macros

	Name	Description
	TCPIP_SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)
	TCPIP_SMTP_TASK_TICK_RATE	SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTP_WRITE_READY_SPACE	the minimum amount of data to ask from the transport/encryption layer when querying the write space
	TCPIP_SMTP_MAX_WRITE_SIZE	the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.

Description

The configuration of the SMPT TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SMTP TCP/IP Stack Library. Based on the selections made, the SMTP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SMTP TCP/IP Stack 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.

TCPIP_SMTP_SERVER_REPLY_TIMEOUT Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_SERVER_REPLY_TIMEOUT (8)
```

Description

How long to wait before assuming the connection has been dropped (default 8 seconds)

TCPIP_SMTP_TASK_TICK_RATE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_TASK_TICK_RATE (55)
```

Description

SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SMTP_WRITE_READY_SPACE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_WRITE_READY_SPACE 150
```

Description

the minimum amount of data to ask from the transport/encryption layer when querying the write space

TCPIP_SMTP_MAX_WRITE_SIZE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_MAX_WRITE_SIZE 512
```

Description

the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.

Building the Library

This section lists the files that are available in the SMTP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/smtp.c`	SMTP 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 SMTP module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.

	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.
	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.
	TCPIP_SMTP_MailSend	Initializes the message sending process.
	TCPIP_SMTP_ClientTask	Standard TCP/IP stack module task function.

b) Data Types and Functions

	Name	Description
	SMTP_CONNECT_ERROR	Connection to SMTP server failed
	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
	SMTP_SUCCESS	Message was successfully sent
	TCPIP_SMTP_CLIENT_MODULE_CONFIG	This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.
	TCPIP_SMTP_CLIENT_MESSAGE	Configures the SMTP client to send a message.

Description

This section describes the Application Programming Interface (API) functions of the SMTP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_SMTP_ArrayPut Function

Writes a series of bytes to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_ArrayPut(uint8_t* Data, uint16_t Len);
```

Returns

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

Description

This function writes a series of bytes to the SMTP client.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written
Len	How many bytes should be written

Function

uint16_t TCPIP_SMTP_ArrayPut(uint8_t* Data, uint16_t Len)

TCPIP_SMTP_Flush Function

Flushes the SMTP socket and forces all data to be sent.

File

smtp.h

C

```
void TCPIP_SMTP_Flush();
```

Returns

None.

Description

This function flushes the SMTP socket and forces all data to be sent.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

void TCPIP_SMTP_Flush(void)

TCPIP_SMTP_IsBusy Function

Determines if the SMTP client is busy.

File

smtp.h

C

```
bool TCPIP_SMTP_IsBusy();
```

Description

Call this function to determine if the SMTP client is busy performing background tasks. This function should be called after any call to TCPIP_SMTP_MailSend, TCPIP_SMTP_PutIsDone to determine if the stack has finished performing its internal tasks. It should also be called prior to any call to TCPIP_SMTP_IsPutReady to verify that the SMTP client has not prematurely disconnected. When this function returns false, the next call should be to TCPIP_SMTP_UsageEnd to release the module and obtain the status code for the operation.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

bool TCPIP_SMTP_IsBusy(void)

TCPIP_SMTP_IsPutReady Function

Determines how much data can be written to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_IsPutReady();
```

Returns

The number of free bytes the SMTP TX FIFO.

Description

Use this function to determine how much data can be written to the SMTP client when generating an on-the-fly message.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call, and an on-the-fly message is being generated. This requires that TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL.

Function

uint16_t TCPIP_SMTP_IsPutReady(void)

TCPIP_SMTP_Put Function

Writes a single byte to the SMTP client.

File

smtp.h

C

```
bool TCPIP_SMTP_Put(char c);
```

Description

This function writes a single byte to the SMTP client.

Remarks

This function is obsolete and will be eventually removed. TCPIP_SMTP_ArrayPut and TCPIP_SMTP_StringPut should be used.

This function cannot be used on an encrypted connection. It is difficult to estimate the amount of TX buffer space needed when transmitting byte by byte, which could cause intermediary write operations to the underlying TCP socket.

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
c	The byte to be written

Function

bool TCPIP_SMTP_Put(char c)

TCPIP_SMTP_PutIsDone Function

Indicates that the on-the-fly message is complete.

File

smtp.h

C

```
void TCPIP_SMTP_PutIsDone();
```

Returns

None.

Description

This function indicates that the on-the-fly message is complete.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call, and the SMTP client is generated an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Function

void TCPIP_SMTP_PutIsDone(void)

TCPIP_SMTP_StringPut Function

Writes a string to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_StringPut(char* Data);
```

Returns

The number of bytes written. If less than the length of Data, then the TX FIFO became full before all bytes could be written.

Description

This function writes a string to the SMTP client.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL.)

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written

Function

uint16_t TCPIP_SMTP_StringPut(char* Data)

TCPIP_SMTP_UsageBegin Function

Requests control of the SMTP client module.

File

smtp.h

C

```
bool TCPIP_SMTP_UsageBegin();
```

Description

Call this function before calling any other SMTP Client APIs. This function obtains a lock on the SMTP Client, which can only be used by one stack application at a time. Once the application is finished with the SMTP client, it must call TCPIP_SMTP_UsageEnd to release control of the module to any other waiting applications.

This function initializes all the SMTP state machines and variables back to their default state.

Preconditions

None.

Function

bool TCPIP_SMTP_UsageBegin(void)

TCPIP_SMTP_UsageEnd Function

Releases control of the SMTP client module.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_UsageEnd();
```

Description

Call this function to release control of the SMTP client module once an application is finished using it. This function releases the lock obtained by TCPIP_SMTP_UsageBegin, and frees the SMTP client to be used by another application.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

uint16_t TCPIP_SMTP_UsageEnd(void)

TCPIP_SMTP_MailSend Function

Initializes the message sending process.

File

smtp.h

C

```
void TCPIP_SMTP_MailSend(TCPIP_SMTP_CLIENT_MESSAGE* smtpClientMessage);
```

Returns

None.

Description

This function starts the state machine that performs the actual transmission of the message. Call this function after all the fields in SMTPClient have been set.

Remarks

The fields pointed by the smtpClientMessage have to be non-volatile until the SMTP send mail process is completed!

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
smtpClientMessage	pointer to a TCPIP_SMTP_CLIENT_MESSAGE structure that configures the message to send

Function

void TCPIP_SMTP_MailSend(TCPIP_SMTP_CLIENT_MESSAGE* smtpClientMessage)

TCPIP_SMTP_ClientTask Function

Standard TCP/IP stack module task function.

File

smtp.h

C

```
void TCPIP_SMTP_ClientTask();
```

Returns

None.

Description

this function performs SMTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

SMTP module should have been initialized.

Function

void TCPIP_SMTP_ClientTask(void)

b) Data Types and Functions

SMTP_CONNECT_ERROR Macro

File

smtp.h

C

```
#define SMTP_CONNECT_ERROR (0x8001u)    // Connection to SMTP server failed
```

Description

Connection to SMTP server failed

SMTP_RESOLVE_ERROR Macro

File

smtp.h

C

```
#define SMTP_RESOLVE_ERROR (0x8000u)    // DNS lookup for SMTP server failed
```

Description

DNS lookup for SMTP server failed

SMTP_SUCCESS Macro

File

smtp.h

C

```
#define SMTP_SUCCESS (0x0000u)    // Message was successfully sent
```

Description

Message was successfully sent

TCPIP_SMTP_CLIENT_MODULE_CONFIG Structure

File

smtp.h

C

```
typedef struct {
```

```
} TCPIP_SMTP_CLIENT_MODULE_CONFIG;
```

Description

This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.

TCPIP_SMTP_CLIENT_MESSAGE Structure

Configures the SMTP client to send a message.

File

smtp.h

C

```
typedef struct {
  char* Server;
  char* Username;
  char* Password;
  char* To;
  char* CC;
  char* BCC;
  char* From;
  char* Subject;
  char* OtherHeaders;
  char* Body;
  bool UseSSL;
  uint16_t ServerPort;
} TCPIP_SMTP_CLIENT_MESSAGE;
```

Description

This structure of pointers configures the SMTP Client to send an e-mail message.

Remarks

When formatting an e-mail address, the SMTP standard format for associating a printable name may be used. This format places the printable name in quotation marks, with the address following in pointed brackets, such as "John Smith" .

Parameters

Parameters	Description
Server	the SMTP server to relay the message through
Username	the user name to use when logging into the SMTP server, if any is required
Password	the password to supply when logging in, if any is required
To	the destination address for this message. May be a comma-separated list of address, and/or formatted.
CC	The CC addresses for this message, if any. May be a comma-separated list of address, and/or formatted.
BCC	The BCC addresses for this message, if any. May be a comma-separated list of address, and/or formatted.
From	The From address for this message. May be formatted.
Subject	The Subject header for this message.
OtherHeaders	Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.
Body	When sending a message from memory, the location of the body of this message in memory. Leave as NULL to build a message on-the-fly.
UseSSL	This flag causes the SMTP client to make a secure connection to the server.
ServerPort	(uint16_t value) Indicates the port on which to connect to the remote SMTP server.

Function

typedef struct TCPIP_SMTP_CLIENT_MESSAGE

Files

Files

Name	Description
smtp.h	Module for Microchip TCP/IP Stack.
smtp_config.h	SMTP configuration file

Description

This section lists the source and header files used by the library.

smtp.h

Module for Microchip TCP/IP Stack.

Functions

	Name	Description
	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.
	TCPIP_SMTP_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
	TCPIP_SMTP_MailSend	Initializes the message sending process.
	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.
	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.

Macros

	Name	Description
	SMTP_CONNECT_ERROR	Connection to SMTP server failed
	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
	SMTP_SUCCESS	Message was successfully sent

Structures

	Name	Description
	TCPIP_SMTP_CLIENT_MESSAGE	Configures the SMTP client to send a message.
	TCPIP_SMTP_CLIENT_MODULE_CONFIG	This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.

Description

Simple Mail Transfer Protocol (SMTP) Client

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

File Name

smtp.h

Company

Microchip Technology Inc.

smtp_config.h

SMTP configuration file

Macros

	Name	Description
	TCPIP_SMTP_MAX_WRITE_SIZE	the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.
	TCPIP_SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)
	TCPIP_SMTP_TASK_TICK_RATE	SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTP_WRITE_READY_SPACE	the minimum amount of data to ask from the transport/encryption layer when querying the write space

Description

Simple Mail Transfer Protocol (SMTP) Configuration file

This file contains the SMTP module configuration options

File Name

smtp_config.h

Company

Microchip Technology Inc.

SMTPC Module

This section describes the TCP/IP Stack Library SMTPC module.

Introduction

TCP/IP Stack Library Simple Mail Transfer Protocol Client (SMTPC) Module for Microchip Microcontrollers

This library provides the API of the SMTPC Client module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SMTPC module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTPC client requires access to a local mail server (such as `mail.yourdomain.com`) for reliable operation. Your MPLAB Harmony or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

Using the Library

This topic describes the basic architecture of the SMTPC TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `smtpc.h`

The interface to the SMTPC TCP/IP Stack library is defined in the `smtpc.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SMTPC TCP/IP Stack library should include `tcpip.h`.

Library File:

The SMTPC TCP/IP Stack library archive (`.a`) file is installed with MPLAB Harmony.

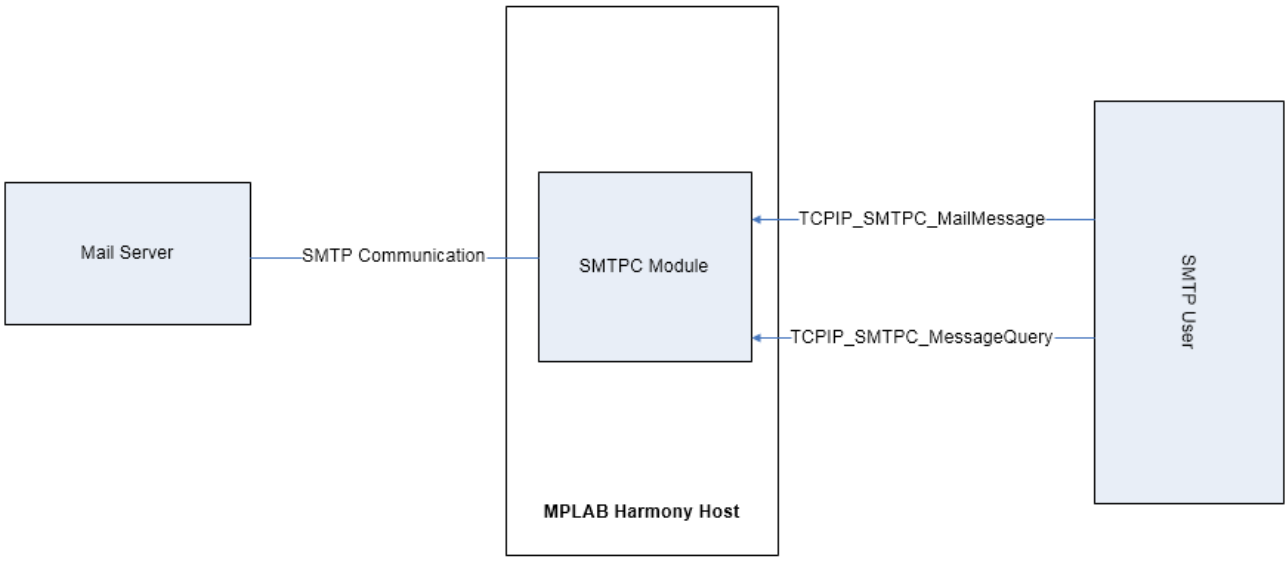
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SMTPC TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SMTPC 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 SMTPC module.

Library Interface Section	Description
Functions	Routines to configure this module
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE	size of a buffer that can hold an email address: user@domain.smth
	TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE	size of a buffer that can hold the 2* username and password
	TCPIP_SMTPC_CLIENT_MESSAGE_DATE	default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers
	TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
	TCPIP_SMTPC_MAIL_CONNECTIONS	number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit
	TCPIP_SMTPC_MAIL_RETRIES	How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)
	TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE	size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.
	TCPIP_SMTPC_SERVER_DATA_TIMEOUT	server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE	size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do
	TCPIP_SMTPC_SERVER_REPLY_TIMEOUT	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
	TCPIP_SMTPC_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.
	TCPIP_SMTPC_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.
	TCPIP_SMTPC_TASK_TICK_RATE	SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT	the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.
	TCPIP_SMTPC_USE_MAIL_COMMAND	Use the sample tcpip console "mail" command for sending email from the console

Description

The configuration of the SMPT TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SMTPC TCP/IP Stack Library. Based on the selections made, the SMTPC TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SMTPC TCP/IP Stack 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.

TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE 80
```

Description

size of a buffer that can hold an email address: user@domain.smth

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here! If buffer not long enough this could generate TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR.

TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE 100
```

Description

size of a buffer that can hold the 2* username and password

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here! If buffer not long enough this could generate TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR.

TCPIP_SMTPC_CLIENT_MESSAGE_DATE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_MESSAGE_DATE "Thu, 14 July 2016 14:55:06 -0600"
```

Description

default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers

TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT 10
```

Description

The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.

TCPIP_SMTPC_MAIL_CONNECTIONS Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_MAIL_CONNECTIONS 3
```

Description

number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit

TCPIP_SMTPC_MAIL_RETRIES Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_MAIL_RETRIES 3
```

Description

How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)

TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE 256
```

Description

size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here!

Lines are sent out as a whole so the line cannot be longer than the TCP socket TX buffer! SMTPC will ensure that this symbol does not exceed the size of the socket TX buffer by allocating a larger TX buffer, if needed

TCPIP_SMTPC_SERVER_DATA_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_DATA_TIMEOUT 60
```

Description

server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough

TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE 512
```

Description

size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do

TCPIP_SMTPC_SERVER_REPLY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_REPLY_TIMEOUT 60
```

Description

general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough

TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT (10 * 60)
```

Description

The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!

TCPIP_SMTPC_SKT_RX_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.

TCPIP_SMTPC_SKT_TX_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.

TCPIP_SMTPC_TASK_TICK_RATE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_TASK_TICK_RATE (55)
```

Description

SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT 10
```

Description

the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.

TCPIP_SMTPC_USE_MAIL_COMMAND Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_USE_MAIL_COMMAND 0
```

Description

Use the sample tcpip console "mail" command for sending email from the console

Building the Library

This section lists the files that are available in the SMTPC module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/smtpc.c`	SMTPC 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 SMTPC module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SMTPC_MailMessage	SMTPC mail message function.
	TCPIP_SMTPC_MessageQuery	SMTPC mail message query function.
	TCPIP_SMTPC_Task	Standard TCP/IP stack module task function.

b) Data Types and Functions

	Name	Description
	TCPIP_SMTPC_ATTACH_BUFFER	Buffer attachment to the mail.
	TCPIP_SMTPC_ATTACH_FILE	File attachment to the mail.
	TCPIP_SMTPC_ATTACH_TYPE	SMTPC attachment type
	TCPIP_SMTPC_ENCODE_TYPE	SMTPC encoding type
	TCPIP_SMTPC_MAIL_FLAGS	Mail message flags
	TCPIP_SMTPC_MAIL_MESSAGE	Mail message descriptor.
	TCPIP_SMTPC_MESSAGE_CALLBACK	Defines a mail message callback
	TCPIP_SMTPC_MESSAGE_HANDLE	Defines a handle to a mail message
	TCPIP_SMTPC_MESSAGE_QUERY	Structure defining a run time query for a mail message
	TCPIP_SMTPC_MESSAGE_REPORT	Structure defining a mail message report
	TCPIP_SMTPC_MESSAGE_RESULT	SMTPC result code
	TCPIP_SMTPC_MESSAGE_STATUS	SMTPC message status
	TCPIP_SMTPC_MESSAGE_WARNING	SMTPC warning type
	TCPIP_SMTPC_MODULE_CONFIG	SMTPC configuration data.
	TCPIP_SMTPC_SERVER_REPLY_CALLBACK	Defines a mail server reply callback

Description

This section describes the Application Programming Interface (API) functions of the SMTPC module.

Refer to each section for a detailed description.

a) Functions

TCPIP_SMTPC_MailMessage Function

SMTPC mail message function.

File

smtpc.h

C

```
TCPIP_SMTPC_MESSAGE_HANDLE TCPIP_SMTPC_MailMessage(const TCPIP_SMTPC_MAIL_MESSAGE* pMailMessage,
TCPIP_SMTPC_MESSAGE_RESULT* pRes);
```

Returns

a valid handle (!= 0) - if mail message scheduled successfully 0 - if call failed

an error code otherwise

Description

This function sends a mail message to the specified server.

Remarks

The returned handle is valid until the final mail notification function is called (TCPIP_SMTPC_MESSAGE_CALLBACK) if exists, or until the mail transaction is completed. After that the messageHandle is no longer valid and should not be used again.

Preconditions

SMTPC module should have been initialized.

Parameters

Parameters	Description
pMailMessage	pointer to a TCPIP_SMTPC_MAIL_MESSAGE describing the mail message to be sent
pRes	address to store the operation result If the call failed, additional info will be reported with this result Can be NULL if not needed.

Function

TCPIP_SMTPC_MESSAGE_HANDLE TCPIP_SMTPC_MailMessage(const TCPIP_SMTPC_MAIL_MESSAGE* pMailMessage, TCPIP_SMTPC_MESSAGE_RESULT* pRes);

TCPIP_SMTPC_MessageQuery Function

SMTPC mail message query function.

File

smtpc.h

C

```
TCPIP_SMTPC_MESSAGE_RESULT TCPIP_SMTPC_MessageQuery(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle,
TCPIP_SMTPC_MESSAGE_QUERY* pQuery);
```

Returns

TCPIP_SMTPC_RES_OK - if message query updated successfully TCPIP_SMTPC_RES_MESSAGE_ERROR - if no such message exists

Description

This function allows the query of a mail message at run time.

Remarks

The message status is for info purposes only. The message status changes as is processed by the SMTPC module.

Once the mail transfer is complete and the corresponding notification function is called, the messageHandle is no longer valid and should not be used again.

Preconditions

SMTPC module should have been initialized.

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
pQuery	pointer to a TCPIP_SMTPC_MESSAGE_QUERY structure that will be updated with the current status of the mail message

Function

TCPIP_SMTPC_MESSAGE_RESULT TCPIP_SMTPC_MessageQuery(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle, TCPIP_SMTPC_MESSAGE_QUERY* pQuery);

TCPIP_SMTPC_Task Function

Standard TCP/IP stack module task function.

File

smtpc.h

C

```
void TCPIP_SMTPC_Task();
```

Returns

None.

Description

this function performs SMTPC module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

SMTPC module should have been initialized.

Function

void TCPIP_SMTPC_Task(void)

b) Data Types and Functions

TCPIP_SMTPC_ATTACH_BUFFER Structure

Buffer attachment to the mail.

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_ATTACH_TYPE attachType;
  TCPIP_SMTPC_ENCODE_TYPE attachEncode;
  const char* attachName;
  const uint8_t* attachBuffer;
  size_t attachSize;
} TCPIP_SMTPC_ATTACH_BUFFER;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE attachType;	type of the contained data
TCPIP_SMTPC_ENCODE_TYPE attachEncode;	preferred encoding
const char* attachName;	file name to be used for the attachment
const uint8_t* attachBuffer;	buffer containing the data
size_t attachSize;	buffer size

Description

Structure: TCPIP_SMTPC_ATTACH_BUFFER

This data structure describes a mail buffer attachment.

Remarks

A buffer attachment is attached to the email just like a file. But data is taken from the persistent user buffer rather than from a file.

If the attach type is application octet stream then the suggested encoding will be overwritten and BASE64 encoding will be used.

The user supplied buffer has to be persistent until the mail completion callback will be called.

TCPIP_SMTPC_ATTACH_FILE Structure

File attachment to the mail.

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_ATTACH_TYPE attachType;
  TCPIP_SMTPC_ENCODE_TYPE attachEncode;
  const char* fileName;
} TCPIP_SMTPC_ATTACH_FILE;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE attachType;	type of the contained data
TCPIP_SMTPC_ENCODE_TYPE attachEncode;	preferred encoding
const char* fileName;	name of the file to be opened

Description

Structure: TCPIP_SMTPC_ATTACH_FILE

This data structure describes a mail file attachment.

Remarks

The specified file will be opened for reading and attached to the email.

If the attach type is application octet stream then the suggested encoding will be overwritten and BASE64 encoding will be used.

The file has to be persistent until the mail completion callback will be called.

If file could not be opened when the mail is sent a warning will be issued and the file will be ignored.

TCPIP_SMTPC_ATTACH_TYPE Enumeration

SMTPC attachment type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_ATTACH_TYPE_UNKNOWN = 0x0000,
  TCPIP_SMTPC_ATTACH_TYPE_APPLICATION = 0x0001,
  TCPIP_SMTPC_ATTACH_TYPE_TEXT = 0x0002,
  TCPIP_SMTPC_ATTACH_TYPE_INLINE = 0x8000
} TCPIP_SMTPC_ATTACH_TYPE;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE_UNKNOWN = 0x0000	unknown attachment type; will default to binary
TCPIP_SMTPC_ATTACH_TYPE_APPLICATION = 0x0001	application octet stream, binary Use this attachment type when sending binary data
TCPIP_SMTPC_ATTACH_TYPE_TEXT = 0x0002	plain text Use this attachment type when sending ASCII text other supported types will be added here
TCPIP_SMTPC_ATTACH_TYPE_INLINE = 0x8000	flag for sending the data as inline instead of attachment

Description

Enumeration: TCPIP_SMTPC_ATTACH_TYPE

Enumeration describing the possible attachment types supported by the SMTPC client.

Remarks

Only one attachment type should be set at a time.

TCPIP_SMTPC_ENCODE_TYPE Enumeration

SMTPC encoding type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_ENCODE_TYPE_UNKNOWN,
  TCPIP_SMTPC_ENCODE_TYPE_7BIT,
  TCPIP_SMTPC_ENCODE_TYPE_BASE64
```

```
} TCPIP_SMTPC_ENCODE_TYPE;
```

Members

Members	Description
TCPIP_SMTPC_ENCODE_TYPE_UNKNOWN	unknown encoding type; defaults to base64
TCPIP_SMTPC_ENCODE_TYPE_7BIT	7bit encoding type: no encoding done, 7 bit ASCII This doesn't do any encoding. Use it for plain text.
TCPIP_SMTPC_ENCODE_TYPE_BASE64	base64 encoding applied Mandatory for binary data; could be used for plain data as well. other supported types will be added here

Description

Enumeration: TCPIP_SMTPC_ENCODE_TYPE

Enumeration describing the possible encoding types supported by the SMTPC client.

Remarks

Normally plain text should be sen using the 7bit encoding type, i.e. plain ASCII. However, selecting base64 encoding has the advantage that the lines are not limited in length and any sequence of characters can be present in the email body without the need of the SMTPC to search and replace the SMTP forbidden characters.

TCPIP_SMTPC_MAIL_FLAGS Enumeration

Mail message flags

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_MAIL_FLAG_USE_IPV6 = 0x0001,
  TCPIP_SMTPC_MAIL_FLAG_BODY_TYPE_BINARY = 0x0002,
  TCPIP_SMTPC_MAIL_FLAG_BODY_ENCODE_ASCII = 0x0004,
  TCPIP_SMTPC_MAIL_FLAG_CONNECT_TLS = 0x0010,
  TCPIP_SMTPC_MAIL_FLAG_SKIP_TLS = 0x0020,
  TCPIP_SMTPC_MAIL_FLAG_FORCE_TLS = 0x0040,
  TCPIP_SMTPC_MAIL_FLAG_AUTH_PLAIN = 0x0100
} TCPIP_SMTPC_MAIL_FLAGS;
```

Members

Members	Description
TCPIP_SMTPC_MAIL_FLAG_USE_IPV6 = 0x0001	use IPv6 connection to the server, rather than default IPv4
TCPIP_SMTPC_MAIL_FLAG_BODY_TYPE_BINARY = 0x0002	mail body is binary rather then standard plain text
TCPIP_SMTPC_MAIL_FLAG_BODY_ENCODE_ASCII = 0x0004	send mail body as plain ASCII encoding rather than default BASE64
TCPIP_SMTPC_MAIL_FLAG_CONNECT_TLS = 0x0010	connect with TLS from the very beginning This flag is useful if the server accepts encrypted connections on port 465, for example Note that port 465 is not really a SMTP port, although supported by many servers!
TCPIP_SMTPC_MAIL_FLAG_SKIP_TLS = 0x0020	skip TLS even if the server supports it most servers won't accept mail if TLS is skipped though
TCPIP_SMTPC_MAIL_FLAG_FORCE_TLS = 0x0040	start TLS even if the server does not indicate support for it
TCPIP_SMTPC_MAIL_FLAG_AUTH_PLAIN = 0x0100	favor PLAIN authentication, if supported by default the LOGIN authentication is selected

Description

Enumeration: TCPIP_SMTPC_MAIL_FLAGS

Enumeration describing the possible mail message flags supported by the SMTPC client.

Remarks

Multiple flags can be set

Only 16 bit flags supported

If no flags are set, then the default settings are used for the mail message.

Normally the mail body is plain text. Using these flags, a binary mail body can be sent.

Normally the mail body is plain text so a standard ASCII encoding is used. Using these flags, a BASE64 encoding for the mail body can be

selected. Note that if the mail body is selected to be binary then the BASE64 encoding is automatically selected.

BASE64 encoding has the advantage that the mail body lines are not limited in length and any sequence of characters can be present in the email body without the need of the SMTPC to search and replace the SMTP forbidden characters.

Normally the connection to the server occurs on a standard SMTP port: 25 or (preferred) 587 as a plain/non-encrypted connection. Once the server replies with TLS support, the encryption is started. Using the flags you can force the connection to be TLS from the very beginning. Or you can avoid engaging in TLS even if the server supports it. Or try to start TLS even if the server didn't advertise support for it.

TCPIP_SMTPC_MAIL_MESSAGE Structure

Mail message descriptor.

File

smtpc.h

C

```
typedef struct {
  const char* from;
  const char* to;
  const char* sender;
  const char* cc;
  const char* bcc;
  const char* date;
  const char* subject;
  const uint8_t* body;
  size_t bodySize;
  int nBuffers;
  const TCPIP_SMTPC_ATTACH_BUFFER* attachBuffers;
  int nFiles;
  const TCPIP_SMTPC_ATTACH_FILE* attachFiles;
  const char* username;
  const char* password;
  const char* smtpServer;
  uint16_t serverPort;
  TCPIP_SMTPC_MAIL_FLAGS messageFlags;
  TCPIP_SMTPC_MESSAGE_CALLBACK messageCallback;
  TCPIP_SMTPC_SERVER_REPLY_CALLBACK replyCallback;
} TCPIP_SMTPC_MAIL_MESSAGE;
```

Members

Members	Description
const char* from;	Mandatory Mail Fields recipient return address
const char* to;	destination address
const char* sender;	Optional Mail Fields if sent on behalf of someone else
const char* cc;	carbon copy destination address
const char* bcc;	blind carbon copy destination address
const char* date;	date for the mail message Should be formatted like: "Wed, 20 July 2016 14:55:06 -0600" if NULL, then the TCPIP_SMTPC_CLIENT_MESSAGE_DATE will be used
const char* subject;	Regular Mail Fields mail subject line
const uint8_t* body;	mail body; main body of the mail
int nBuffers;	Mail Attachments number of attached buffers
const TCPIP_SMTPC_ATTACH_BUFFER* attachBuffers;	array of buffer attachments
int nFiles;	number of attached files
const TCPIP_SMTPC_ATTACH_FILE* attachFiles;	array of file attachments
const char* username;	Mail Authentication username to be presented when the mail server requires authentication
const char* password;	associated password to be presented when the mail server requires authentication
const char* smtpServer;	Mail Server address or name of a server to send the mail to (smtp.gmail.com, 74.125.23.109, etc.)

uint16_t serverPort;	port on which the SMTP server listens on Standard SMTP ports are 25 or 587 For servers supporting TLS (gmail, yahoo, etc.), the recommended port is 587
TCPIP_SMTPC_MAIL_FLAGS messageFlags;	Additional message flags
TCPIP_SMTPC_MESSAGE_CALLBACK messageCallback;	Mail Notifications message completion notification could be NULL if not needed
TCPIP_SMTPC_SERVER_REPLY_CALLBACK replyCallback;	server reply notification could be NULL if not needed

Description

Structure: TCPIP_SMTPC_MAIL_MESSAGE

This data structure describes a mail message

Remarks

SMTPC does NOT make private copies of the strings, buffers, etc. presented in a TCPIP_SMTPC_MAIL_MESSAGE data structure. All fields that are pointers to strings, buffers, files names, etc. MUST be persistent until the mail complete notification is received.

The TCPIP_SMTPC_MAIL_MESSAGE data structure itself does not need to be persistent.

The mail body is not a char* but uint8_t* to allow sending binary data as well as part of the mail body. This requires though that the bodySize member is present.

If some file could not be opened when the mail is sent a warning will be issued and the file will be ignored.

Currently the fields: "sender", "cc" and "bcc" are not used. They will be eventually added.

TCPIP_SMTPC_MESSAGE_CALLBACK Type

Defines a mail message callback

File

smtpc.h

C

```
typedef void (* TCPIP_SMTPC_MESSAGE_CALLBACK)(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle, const
TCPIP_SMTPC_MESSAGE_REPORT* pMailReport);
```

Description

Type: TCPIP_SMTPC_MESSAGE_CALLBACK

Data type describing a function that will be called by the SMTPC module upon completion of the mail message.

Remarks

A message handle is used rather than the user's TCPIP_SMTPC_MAIL_MESSAGE. This allows sending multiple copies of the message without waiting for a previous copy to be done.

This notification can be called multiple times if the delivery attempt failed for a transient reson. The number of left retries tells if another attempt will be retried.

Once this final notification function is called (left retries == 0), the SMTPC no longer uses the TCPIP_SMTPC_MAIL_MESSAGE data (unless there are other copies of this message in transit). This messageHandle is no longer valid and should not be used again.

The pMailReport points to a SMTPC owned read only data structure. It should be used only to inspect the message delivery result.

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
pMailReport	pointer to a TCPIP_SMTPC_MESSAGE_REPORT data structure containing the
result of the message delivery	messageRes - the result of the message
errorStat	error status
messageWarn	warnings
leftRetries	number of times the delivery will be re-attempted.

TCPIP_SMTPC_MESSAGE_HANDLE Type

Defines a handle to a mail message

File

smtpc.h

C

```
typedef const void* TCPIP_SMTPC_MESSAGE_HANDLE;
```

Description

Type: TCPIP_SMTPC_MESSAGE_HANDLE

Data type describing a handle that identifies a mail message that's scheduled for transmission. This handle is to be used for retrieving the message status. This handle will be part of the callback that the SMTP client will use to notify the user of the mail completion.

A valid handle is != 0.

Remarks

None

TCPIP_SMTPC_MESSAGE_QUERY Structure

Structure defining a run time query for a mail message

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_MESSAGE_STATUS messageStat;
  TCPIP_SMTPC_MESSAGE_WARNING messageWarn;
  int messageRetries;
  NET_PRES_SKT_HANDLE_T messageSkt;
} TCPIP_SMTPC_MESSAGE_QUERY;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_STATUS messageStat;	the current status of the mail message
TCPIP_SMTPC_MESSAGE_WARNING messageWarn;	the current warnings encountered during the message transmission
int messageRetries;	the current number of retries for this message
NET_PRES_SKT_HANDLE_T messageSkt;	the socket associated with this message

Description

Structure: TCPIP_SMTPC_MESSAGE_QUERY

This data structure describes a mail message query at run time.

Remarks

The message status is for info purposes only. The message status changes as is processed by the SMTPC module.

TCPIP_SMTPC_MESSAGE_REPORT Structure

Structure defining a mail message report

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_MESSAGE_RESULT messageRes;
  TCPIP_SMTPC_MESSAGE_STATUS errorStat;
  TCPIP_SMTPC_MESSAGE_WARNING messageWarn;
  int leftRetries;
} TCPIP_SMTPC_MESSAGE_REPORT;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_RESULT messageRes;	the result of the message TCPIP_SMTPC_RES_OK if the message was successfully transmited else some error were encountered
TCPIP_SMTPC_MESSAGE_STATUS errorStat;	the mail message status in which the error occurred, if any Else the TCPIP_SMTPC_MESSAGE_STAT_NONE will be used
TCPIP_SMTPC_MESSAGE_WARNING messageWarn;	warnings encountered during the message transmission, if any
int leftRetries;	the number of retries that will still be attempted if error was a transient, not fatal

Description

Structure: TCPIP_SMTPC_MESSAGE_REPORT

This data structure describes a mail message report. It is used by the SMTPC to report an mail delivery result.

Remarks

None

TCPIP_SMTPC_MESSAGE_RESULT Enumeration

SMTPC result code

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_RES_OK = 0,
  TCPIP_SMTPC_RES_PENDING = 1,
  TCPIP_SMTPC_RES_MESSAGE_ERROR = -1,
  TCPIP_SMTPC_RES_MESSAGE_SERVER_ERROR = -2,
  TCPIP_SMTPC_RES_MESSAGE_RCPT_ERROR = -3,
  TCPIP_SMTPC_RES_MESSAGE_BUFFER_ERROR = -4,
  TCPIP_SMTPC_RES_MESSAGE_FILE_ERROR = -5,
  TCPIP_SMTPC_RES_MESSAGE_AUTH_REQUIRED = -6,
  TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR = -7,
  TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR = -8,
  TCPIP_SMTPC_RES_MAIL_BUSY = -9,
  TCPIP_SMTPC_RES_DNS_ERROR = -10,
  TCPIP_SMTPC_RES_SKT_OPEN_ERROR = -11,
  TCPIP_SMTPC_RES_SKT_BIND_ERROR = -12,
  TCPIP_SMTPC_RES_SKT_CONNECT_TMO = -13,
  TCPIP_SMTPC_RES_SKT_TLS_ERROR = -14,
  TCPIP_SMTPC_RES_SERVER_TMO = -15,
  TCPIP_SMTPC_RES_CONNECTION_REJECT = -16,
  TCPIP_SMTPC_RES_CONNECTION_CLOSE = -17,
  TCPIP_SMTPC_RES_HELLO_REJECT = -18,
  TCPIP_SMTPC_RES_AUTH_UNKNOWN = -19,
  TCPIP_SMTPC_RES_AUTH_LOGIN_REJECT = -20,
  TCPIP_SMTPC_RES_AUTH_LOGIN_SERVER_ERROR = -21,
  TCPIP_SMTPC_RES_AUTH_REJECT = -22,
  TCPIP_SMTPC_RES_TLS_REJECT = -23,
  TCPIP_SMTPC_RES_TLS_FAILED = -24,
  TCPIP_SMTPC_RES_TLS_TMO = -25,
  TCPIP_SMTPC_RES_MAIL_FROM_REJECT = -26,
  TCPIP_SMTPC_RES_MAIL_RCPT_REJECT = -27,
  TCPIP_SMTPC_RES_MAIL_DATA_REJECT = -28,
  TCPIP_SMTPC_RES_MAIL_BODY_REJECT = -29,
  TCPIP_SMTPC_RES_INITIALIZE_ERROR = -40,
  TCPIP_SMTPC_RES_INTERNAL_ERROR = -41
} TCPIP_SMTPC_MESSAGE_RESULT;
```

Members

Members	Description
TCPIP_SMTPC_RES_OK = 0	OK, operation successful
TCPIP_SMTPC_RES_PENDING = 1	operation in progress
TCPIP_SMTPC_RES_MESSAGE_ERROR = -1	mail message error

TCPIP_SMTPC_RES_MESSAGE_SERVER_ERROR = -2	message indicated wrong mail server
TCPIP_SMTPC_RES_MESSAGE_RCPT_ERROR = -3	message mail recipient error: from, to, etc.
TCPIP_SMTPC_RES_MESSAGE_BUFFER_ERROR = -4	attachment buffer error
TCPIP_SMTPC_RES_MESSAGE_FILE_ERROR = -5	attachment file error
TCPIP_SMTPC_RES_MESSAGE_AUTH_REQUIRED = -6	server requires authentication but username or password haven't been provided
TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR = -7	provided credentials are too long, buffer overflow
TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR = -8	email address too long, buffer overflow
TCPIP_SMTPC_RES_MAIL_BUSY = -9	all mail connections are busy; try later
TCPIP_SMTPC_RES_DNS_ERROR = -10	failure to resolve server name
TCPIP_SMTPC_RES_SKT_OPEN_ERROR = -11	failure to open a communication socket
TCPIP_SMTPC_RES_SKT_BIND_ERROR = -12	failure to bind a socket to the mail server
TCPIP_SMTPC_RES_SKT_CONNECT_TMO = -13	connection to mail server timeout
TCPIP_SMTPC_RES_SKT_TLS_ERROR = -14	TLS is required but failed to start TLS on the communication socket
TCPIP_SMTPC_RES_SERVER_TMO = -15	server timeout
TCPIP_SMTPC_RES_CONNECTION_REJECT = -16	server rejected the connection
TCPIP_SMTPC_RES_CONNECTION_CLOSE = -17	server closed the connection
TCPIP_SMTPC_RES_HELLO_REJECT = -18	server rejected the hello greeting
TCPIP_SMTPC_RES_AUTH_UNKNOWN = -19	server requires authentication mechanism unsupported by SMTPC Currently LOGIN and PLAIN authentications are supported
TCPIP_SMTPC_RES_AUTH_LOGIN_REJECT = -20	server rejected the login authentication request
TCPIP_SMTPC_RES_AUTH_LOGIN_SERVER_ERROR = -21	unexpected server reply to login authentication request
TCPIP_SMTPC_RES_AUTH_REJECT = -22	server rejected the supplied authentication
TCPIP_SMTPC_RES_TLS_REJECT = -23	server rejected the TLS start
TCPIP_SMTPC_RES_TLS_FAILED = -24	TLS session negotiation failed
TCPIP_SMTPC_RES_TLS_TMO = -25	TLS session timeout
TCPIP_SMTPC_RES_MAIL_FROM_REJECT = -26	server rejected the "from" address
TCPIP_SMTPC_RES_MAIL_RCPT_REJECT = -27	server rejected the "recipient" address
TCPIP_SMTPC_RES_MAIL_DATA_REJECT = -28	server rejected the "data" field
TCPIP_SMTPC_RES_MAIL_BODY_REJECT = -29	server rejected the mail body
TCPIP_SMTPC_RES_INITIALIZE_ERROR = -40	module is not properly initialized, API call is invalid
TCPIP_SMTPC_RES_INTERNAL_ERROR = -41	internal failure, should not happen

Description

Enumeration: TCPIP_SMTPC_MESSAGE_RESULT

Enumeration describing the possible result codes returned by the SMTPC client.

Remarks

None

TCPIP_SMTPC_MESSAGE_STATUS Enumeration

SMTPC message status

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_MESSAGE_STAT_NONE,
  TCPIP_SMTPC_MESSAGE_STAT_DNS,
  TCPIP_SMTPC_MESSAGE_STAT_CONNECT,
  TCPIP_SMTPC_MESSAGE_STAT_HELLO,
  TCPIP_SMTPC_MESSAGE_STAT_TLS,
```

```
  TCPIP_SMTPC_MESSAGE_STAT_AUTH,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_ENVELOPE,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_BODY,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_BUFFERS,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_FILES,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_ACKNOWLEDGE,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_REPORT,
  TCPIP_SMTPC_MESSAGE_STAT_CLOSING,
  TCPIP_SMTPC_MESSAGE_STAT_WAIT_RETRY
} TCPIP_SMTPC_MESSAGE_STATUS;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_STAT_NONE	there is no transaction started
TCPIP_SMTPC_MESSAGE_STAT_DNS	doing the DNS resolution
TCPIP_SMTPC_MESSAGE_STAT_CONNECT	connecting to the server
TCPIP_SMTPC_MESSAGE_STAT_HELLO	saying hello to the server
TCPIP_SMTPC_MESSAGE_STAT_TLS	starting the TLS communication with the server
TCPIP_SMTPC_MESSAGE_STAT_AUTH	authenticating to the server
TCPIP_SMTPC_MESSAGE_STAT_MAIL_ENVELOPE	sending mail envelope
TCPIP_SMTPC_MESSAGE_STAT_MAIL_BODY	sending mail body
TCPIP_SMTPC_MESSAGE_STAT_MAIL_BUFFERS	sending mail buffers
TCPIP_SMTPC_MESSAGE_STAT_MAIL_FILES	sending mail files
TCPIP_SMTPC_MESSAGE_STAT_MAIL_ACKNOWLEDGE	waiting for the mail acknowledge
TCPIP_SMTPC_MESSAGE_STAT_MAIL_REPORT	reporting the mail outcome
TCPIP_SMTPC_MESSAGE_STAT_CLOSING	closing the mail transaction
TCPIP_SMTPC_MESSAGE_STAT_WAIT_RETRY	waiting for a retry

Description

Enumeration: TCPIP_SMTPC_MESSAGE_STATUS

Enumeration describing the possible states of a mail message during a mail transmission. The message progresses through various states until complete mail transfer or error

Remarks

TCPIP_SMTPC_MESSAGE_STAT_HELLO will occur twice if TLS is supported by the server. Once the connection is secured, another HELLO message will be sent to the server.

TCPIP_SMTPC_MESSAGE_WARNING Enumeration

SMTPC warning type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_WARN_REPLY_BUFFER_SMALL = 0x0001,
  TCPIP_SMTPC_WARN_WRONG_FORMAT = 0x0002,
  TCPIP_SMTPC_WARN_UNEXPECTED_REPLY = 0x0004,
  TCPIP_SMTPC_WARN_FILE_NOT_FOUND = 0x0008,
  TCPIP_SMTPC_WARN_AUTH_LOGIN_LEN = 0x0010
} TCPIP_SMTPC_MESSAGE_WARNING;
```

Members

Members	Description
TCPIP_SMTPC_WARN_REPLY_BUFFER_SMALL = 0x0001	the reply buffer allocated for the server replies is too small server reply exceeded the size of the buffer it should be increased
TCPIP_SMTPC_WARN_WRONG_FORMAT = 0x0002	server reply was not properly formatted
TCPIP_SMTPC_WARN_UNEXPECTED_REPLY = 0x0004	unexpected server replied received
TCPIP_SMTPC_WARN_FILE_NOT_FOUND = 0x0008	file attachment not found; ignored

TCPIP_SMTPC_WARN_AUTH_LOGIN_LEN = 0x0010	provided credentials for LOGIN authentication are too long and were truncated (buffer overflow) this most likely will result in a server authentication rejection

Description

Enumeration: TCPIP_SMTPC_MESSAGE_WARNING

Enumeration describing the possible warning types occurring during a mail transmission. These events are not fatal and SMTPC client will try to complete the mail transfer.

Remarks

Multiple warnings could be set during a mail transmission

TCPIP_SMTPC_MODULE_CONFIG Structure

SMTPC configuration data.

File

smtpc.h

C

```
typedef struct {
  int nMailConnections;
  uint32_t serverReplyTmo;
  uint32_t serverDataTmo;
  uint32_t tlsHandshakeTmo;
  int nMailRetries;
  uint32_t serverRetryTmo;
  uint32_t smtpcRetryTmo;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
} TCPIP_SMTPC_MODULE_CONFIG;
```

Members

Members	Description
int nMailConnections;	number of mail connections to be created each mail message to be sent requires a mail connection adjust based on the number of simultaneous mail messages in transit
uint32_t serverReplyTmo;	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min
uint32_t serverDataTmo;	server response timeout to acknowledge a received mail body/data, seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min
uint32_t tlsHandshakeTmo;	secure connection establishment timeout, seconds usually few seconds
int nMailRetries;	how many times to retry sending a mail message Retries occur only for server reported transient errors
uint32_t serverRetryTmo;	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
uint32_t smtpcRetryTmo;	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default

Description

Structure: TCPIP_SMTPC_MODULE_CONFIG

This data structure describes the SMTPC configuration data. It is presented to the module as part of the module initialization.

Remarks

None

TCPIP_SMTPC_SERVER_REPLY_CALLBACK Type

Defines a mail server reply callback

File

smtpc.h

C

```
typedef void (* TCPIP_SMTPC_SERVER_REPLY_CALLBACK)(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle,
TCPIP_SMTPC_MESSAGE_STATUS currStat, const char* serverReply);
```

Description

Type: TCPIP_SMTPC_SERVER_REPLY_CALLBACK

Data type describing a function that will be called by the SMTPC module upon receiving a message from the mail server. SMTPC will parse and process the server messages anyway but it can optionally pass the server replies to the user.

Remarks

None

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
currStat	the current status for the mail message
serverReply	the reply sent by the server

Files

Files

Name	Description
smtpc.h	Module for Microchip TCP/IP Stack.
smtpc_config.h	SMTPC configuration file

Description

This section lists the source and header files used by the library.

smtpc.h

Module for Microchip TCP/IP Stack.

Enumerations

	Name	Description
	TCPIP_SMTPC_ATTACH_TYPE	SMTPC attachment type
	TCPIP_SMTPC_ENCODE_TYPE	SMTPC encoding type
	TCPIP_SMTPC_MAIL_FLAGS	Mail message flags
	TCPIP_SMTPC_MESSAGE_RESULT	SMTPC result code
	TCPIP_SMTPC_MESSAGE_STATUS	SMTPC message status
	TCPIP_SMTPC_MESSAGE_WARNING	SMTPC warning type

Functions

	Name	Description
	TCPIP_SMTPC_MailMessage	SMTPC mail message function.
	TCPIP_SMTPC_MessageQuery	SMTPC mail message query function.
	TCPIP_SMTPC_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_SMTPC_ATTACH_BUFFER	Buffer attachment to the mail.
	TCPIP_SMTPC_ATTACH_FILE	File attachment to the mail.
	TCPIP_SMTPC_MAIL_MESSAGE	Mail message descriptor.
	TCPIP_SMTPC_MESSAGE_QUERY	Structure defining a run time query for a mail message
	TCPIP_SMTPC_MESSAGE_REPORT	Structure defining a mail message report
	TCPIP_SMTPC_MODULE_CONFIG	SMTPC configuration data.

Types

	Name	Description
	TCPIP_SMTPC_MESSAGE_CALLBACK	Defines a mail message callback
	TCPIP_SMTPC_MESSAGE_HANDLE	Defines a handle to a mail message
	TCPIP_SMTPC_SERVER_REPLY_CALLBACK	Defines a mail server reply callback

Description

Simple Mail Transfer Protocol Client - SMTPC

The SMTPC client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

File Name

smtpc.h

Company

Microchip Technology Inc.

smtpc_config.h

SMTPC configuration file

Macros

	Name	Description
	TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE	size of a buffer that can hold an email address: user@domain.smth
	TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE	size of a buffer that can hold the 2* username and password
	TCPIP_SMTPC_CLIENT_MESSAGE_DATE	default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers
	TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
	TCPIP_SMTPC_MAIL_CONNECTIONS	number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit
	TCPIP_SMTPC_MAIL_RETRIES	How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)
	TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE	size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.
	TCPIP_SMTPC_SERVER_DATA_TIMEOUT	server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE	size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do
	TCPIP_SMTPC_SERVER_REPLY_TIMEOUT	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
	TCPIP_SMTPC_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.
	TCPIP_SMTPC_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.

	TCPIP_SMTPC_TASK_TICK_RATE	SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT	the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.
	TCPIP_SMTPC_USE_MAIL_COMMAND	Use the sample tcpip console "mail" command for sending email from the console

Description

Simple Mail Transfer Protocol Client (SMTPC) Configuration file

This file contains the SMTPC module configuration options

File Name

smtpc_config.h

Company

Microchip Technology Inc.

SNMP Module

This section describes the TCP/IP Stack Library SNMP module.

Introduction

TCP/IP Stack Library Simple Network Management Protocol (SNMP) Module for Microchip Microcontrollers

This library provides the API of the SNMP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

SNMP is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite.

SNMP is an Internet protocol that was originally designed to manage different network devices, such as file servers, hubs, routers and so on. It can also be used to manage and control an ever increasing number of small embedded systems connected to one another over any IP network. Systems can communicate with each other using SNMP to transfer control and status information, creating a truly distributed system.

SNMP is used in a variety of applications where remote monitoring and controlling of the network node is desired, such as a network printer, online Uninterrupted Power Supply (UPS), security cameras, home and industrial appliances monitor and control, automatic energy meter readings, etc.

Unlike more familiar human-oriented protocols, like HTTP, SNMP is considered a machine-to-machine protocol.

Note: The related application note, AN870 *"SNMP V2c Agent for Microchip TCP/IP Stack"* (DS8000870) is available for download from the Microchip web site at: http://ww1.microchip.com/downloads/en/AppNotes/00870b.pdf

Using the Library

This topic describes the basic architecture of the SNMP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `snmp.h`

The interface to the SNMP TCP/IP Stack library is defined in the `snmp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SNMP TCP/IP Stack library should include `tcpip.h`.

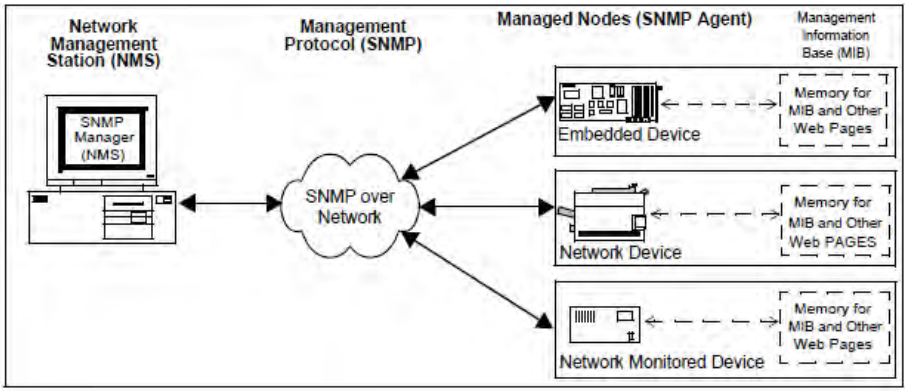
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SNMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNMP 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 SNMP module.

Library Interface Section	Description
SNMP Module Functions	Routines to configure this module
SNMPv3 Module Functions	Routines to configure this module
SNMP Application Functions	Routines to configure an application
SNMPv3 Application Functions	Routines to configure an application
SNMP Data Types and Constants	This section provides various definitions describing this API
SNMPv3 Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_SNMP_BIB_FILE_NAME	The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.
	TCPIP_SNMP_COMMUNITY_MAX_LEN	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
	TCPIP_SNMP_MAX_COMMUNITY_SUPPORT	Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	TCPIP_SNMP_MAX_MSG_SIZE	The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16
	TCPIP_SNMP_MAX_NON_REC_ID_OID	Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it... more
	TCPIP_SNMP_NOTIFY_COMMUNITY_LEN	Maximum length for SNMP Trap community name
	TCPIP_SNMP_OID_MAX_LEN	Maximum length for the OID String. Change this to match your OID string length.
	TCPIP_SNMP_TASK_PROCESS_RATE	SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN	The maximum size of TRAP community string length
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	Trap information. This macro will be used to avoid SNMP OID memory buffer corruption
	TCPIP_SNMP_TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Authentication Localized password key length size
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Privacy Password key length size
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN	Maximum size for SNMPv3 User Security Name length.
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	TCPIP_SNMPV3_USM_MAX_USER	Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Description

The configuration of the SNMP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SNMP TCP/IP Stack Library. Based on the selections made, the SNMP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SNMP TCP/IP Stack 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.

TCPIP_SNMP_BIB_FILE_NAME Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_BIB_FILE_NAME "snmp.bib"
```

Description

The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.

TCPIP_SNMP_COMMUNITY_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_COMMUNITY_MAX_LEN (8u)
```

Description

This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.

TCPIP_SNMP_MAX_COMMUNITY_SUPPORT Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_COMMUNITY_SUPPORT (3u)
```

Description

Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.

TCPIP_SNMP_MAX_MSG_SIZE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_MSG_SIZE 480
```

Description

The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16

TCPIP_SNMP_MAX_NON_REC_ID_OID Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_NON_REC_ID_OID 3
```

Description

Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it also includes MODULE-IDENTITY (number 1)after ENTERPRISE-ID.

TCPIP_SNMP_NOTIFY_COMMUNITY_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_NOTIFY_COMMUNITY_LEN (TCPIP_SNMP_COMMUNITY_MAX_LEN)
```

Description

Maximum length for SNMP Trap community name

TCPIP_SNMP_OID_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_OID_MAX_LEN (18)
```

Description

Maximum length for the OID String. Change this to match your OID string length.

TCPIP_SNMP_TASK_PROCESS_RATE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TASK_PROCESS_RATE (200)
```

Description

SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN (TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE+1)
```

Description

The maximum size of TRAP community string length

TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE (8)
```

Description

Trap information. This macro will be used to avoid SNMP OID memory buffer corruption

TCPIP_SNMP_TRAP_TABLE_SIZE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_TABLE_SIZE (2)
```

Description

This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Authentication Localized password key length size

TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
(TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 authentication localized Key length for memory validation

TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Privacy Password key length size

TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
(TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 privacy key length size for memory validation

TCPIP_SNMPV3_USER_SECURITY_NAME_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USER_SECURITY_NAME_LEN (16)
```

Description

Maximum size for SNMPv3 User Security Name length.

TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE (TCPIP_SNMPV3_USER_SECURITY_NAME_LEN+1)
```

Description

User security name length for memory validation

TCPIP_SNMPV3_USM_MAX_USER Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USM_MAX_USER 3
```

Description

Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Building the Library

This section lists the files that are available in the SNMP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/snmp.c`	SNMP 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 SNMP module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- UDP Module

SNMP Server (Agent)

This section describes the SNMP Server Agent.

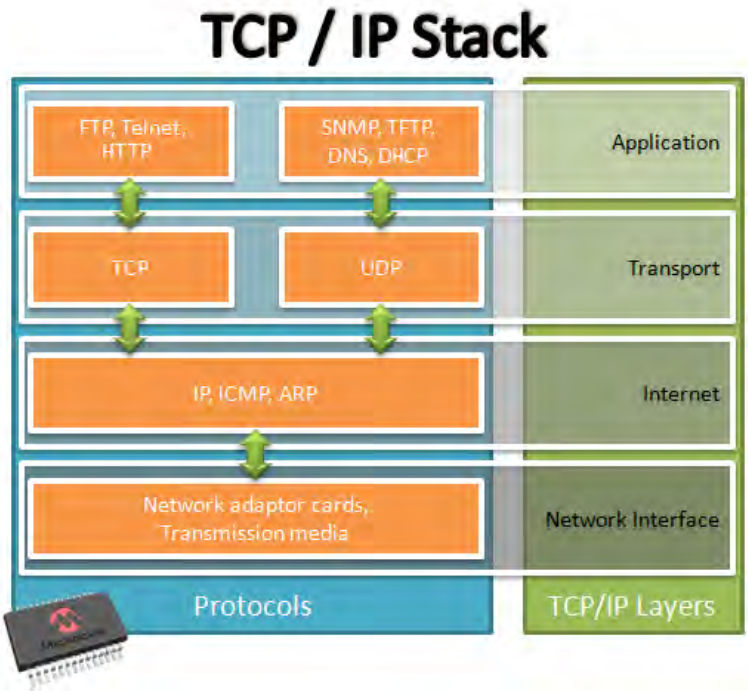
Introduction

The Simple Network Management Protocol (SNMP) is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite.

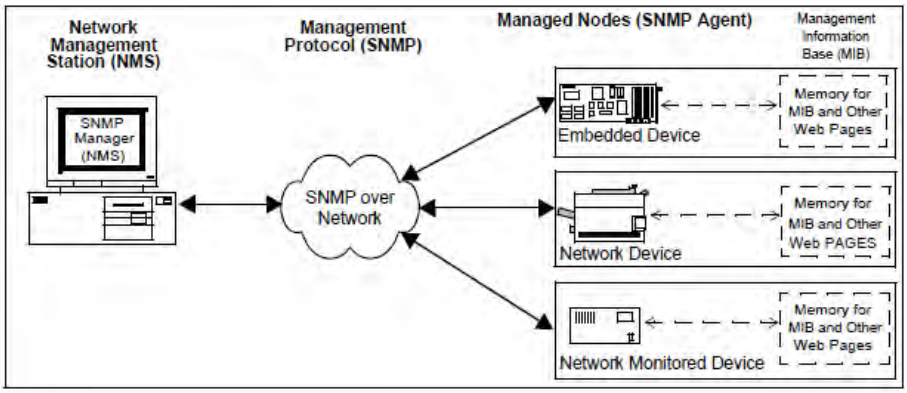
Description

SNMP is an Internet protocol that was originally designed to manage different network devices, such as file servers, hubs, routers, and so on. It can also be used to manage and control an ever increasing number of small embedded systems connected to one another over any IP network. Systems can communicate with each other using SNMP to transfer control and status information, creating a truly distributed system.

Location of the SNMP Stack in the TCP/IP Protocol Stack



Overview of the SNMP Model



The Microchip SNMP Server is a multi-lingual implementation, which supports SNMPv1, V2c, and V3 server features simultaneously. The SNMP Server is implemented to address the requirements of embedded applications and works with both IPv4 and IPv6 addresses. SNMPv1 and V2c are enabled by TCPIP_STACK_USE_SNMP_SERVER. SNMPv3 support is enabled by TCPIP_STACK_USE_SNMPV3_SERVER. Since the SNMPv3 stack requires the support of SNMPv1 and SNMPv2c, TCPIP_STACK_USE_SNMPV3_SERVER should be enabled with TCPIP_STACK_USE_SNMP_SERVER.

Note: Currently, the SNMP agent will be able to communicate with one manager at a time, and it will be connected to either an IPv4 address or an IPv6 address.

V2c

V2c is implemented with support for the configuration of multiple community names and the community names can be configured through the HTTP web interface. An access-restricted web page is provided with the demonstration application to allow dynamic configuration of SNMP communities.

SNMPv3

SNMPv3 RFC specifies different types of access mechanisms, user security model (USM), authentication and privacy protocols. The Microchip SNMPv3 Server is implemented with support for the AES 128 CFB 128 and DES-CBC privacy protocols, and the MD5 and SHA1 message authentication protocols. The demonstration implementation of the server is configured with three types of user names with respective authentication and privacy credentials and authentication types. These credentials and other user information are stored in the global array. The user of the SNMPv3 stack can decide on the number of user names in the User's database to be stored with the SNMPv3 Server. According to the SNMPv3 recommendation, the SNMPv3 Server should not be configured with the authentication and privacy passwords. Instead, it could be configured with the respective localized keys of the password. The Microchip SNMPv3 agent is provided with the password information in the database for "Getting Started" and for understanding purposes only. It is recommended that the SNMPv3 stack be modified to restrict access to the password OIDs declared in the user database.

Note: Even though SNMPv3 also requires SNMPv1 and SNMPv2c, a layer in the SNMP Stack will prevent access to the variables that should be secured by SNMPv3. SNMP variables are structures in a tree in the MIB. Access to parts of this tree is determined by version. For example, SYSTEM-type variables can be accessed regardless of the SNMP version, while SNMPv2c requests can access part of the tree, and authenticated SNMPv3 requests can access the complete tree.

MIB Files

Provides information the Management Information Base (MIB) files.

Description

SNMP describes the hierarchal storage of management objects (referred to with object IDs or OIDs) with MIB files. The Microchip SNMP server demonstration includes two MIB files:

- `mchip.mib` - This is an Abstract Syntax Notation One (ASN.1) formatted MIB file containing information about the variables used in the demonstration
- `snmp.mib` - This is a custom-formatted file that can be parsed to create Web page and header resources that can be accessed with a PIC microcontroller

The TCP/IP Stack includes the mib2bib utility, which will compile the custom Microchip MIB script (`snmp.mib`) to generate two files named snmp.bib and `mib.h`. The snmp.bib file is a compressed record of management objects that will be stored with web pages and the `mib.h` file contains C defines for each OID. These files are included in the appropriate directories for the TCP/IP Demonstration Applications, but for a custom application you must copy snmp.bib to your web page directory, copy `mib.h` to your application directory and include it in your project, rebuild your project, and then rebuild and reupload your web page. This will bundle the BIB file into your web page image, which will allow the SNMP agent to search for the required variable information with the MPFS file system.

MIB Browsers

Provides information the available MIB browsers.

Description

Several SNMP MIB browsers are available. Users can also install a customized MIB browser specific to their application.

This documentation describes how to use the iREASONING Networks MIB Browser to run the TCP/IP SNMP demonstration applications. The MIB Browser can be obtained from: http://www.ireasoning.com/downloadmibbrowserlicense.shtml. The MIB script upload, the MIB tree structure display, and the SNMP query mechanism procedures vary from browser to browser.

Important! The use of a MIB browser or other third-party products may require that users review and agree to the terms of a license. Microchip's reference to the iREASONING Networks MIB Browser is for the users' convenience. It is the user's responsibility to obtain information about, and comply with the terms of, any applicable licenses.

Refer to the Microchip application note, AN870 "SNMP V2c Agent for Microchip TCP/IP Stack" (DS00000870) for more details on the MIB scripts, community names, and demonstration SNMP MIB variable tree structure.

The ASN.1 format `mchip.mib` file is defined with a private variable tree structure for the MIB variables. Also the `mchip.mib` is added with the number of OIDs that could be accessed only with SNMPv3 request. The browser can access every variable in the MIB database provided the community name matches. The access to the MIB variables is restricted to the type of the request. The RFC1213 MIB variables could be accessed with SNMPv2c/v3 request. But the SNMP-FRAMEWORK-MIB.mib variables could only be accessed with a SNMPv3 request if the credentials are matched and the message is authenticated. To modify these MIB variables, corresponding changes must be made to both MIB scripts (`snmp.mib` and `mchip.mib`).

- For SNMP V2c services , The V2c agent is configured by default with three Read communities ("public", "read", "") and three Write communities ("private","write","public")
- At run-time, the community names can be dynamically configured using the HTTP interface for SNMP community name configuration
- If the V2c agent receives an SNMP request with an unknown community name, the agent will generate an Authentication trap

The V2c agent's multiple community support feature enables the user application to provide limited access to the requesting browser based on the community name used by the browser to access the MIB database variables of the agent.

For SNMPv3 services:

Type	USER 1	USER 2	USER 3
USM User	microchip	SnmpAdmin	root
Security Level	auth, priv	auth, no priv	no auth, no priv
Auth Algorithm	MD5	SHA1	N/A
Auth Password	auth12345	ChandlerUS	N/A
Privacy Algorithm	AES	N/A	N/A
Privacy Password	priv12345	N/A	N/A

If SNMPv3 services are required, SNMPv3 browser is required to be configured with the user name, authentication and privacy password, message authentication hash type, privacy protocol type. The SNMP server would respond only if one of the user credentials and user security parameters in the following table is configured at the manager. The previous table is stored in the global structure with the SNMPv3 server stack. The SNMPv3 server would only respond if the request credentials of the MIB browser matches to that of the stored user data base of the SNMP server.

The Microchip SNMPv3 stack does support only one Context Engine ID with the server. Leave the "Context Name" option in the "Advanced" tab empty. It is ignored on the server.

According to the user and the auth and privacy protocols configured with the SNMP browser, the UDP authenticated and encrypted message would be exchanged between server and the client.

- If the USER 1 values, as shown in the table, are configured in the MIB browser, the data exchange between the client and server is encrypted and authenticated. The PDU could be captured in the Ethernet packet sniffer, such as WireShark, and examined. As the data is encrypted and authenticated, the data integrity and the privacy is achieved.
- If USER 2 values, as shown in the table, are configured in the MIB browser, the data exchange between client and server is authenticated. The data integrity would be checked once the data is received at either end. The message authentication mechanism protects from the possible data sniffing and modification threat, and also guarantees that the data is received from the authenticated and guaranteed source.
- If USER 3 values, as shown in the table, are configured in the MIB browser, the data exchange between client and server is neither authenticated nor encrypted
- Considering the three USER configurations, if the SNMP server is to be accessed over WAN, in the Internet cloud, the data should be encrypted and authenticated to have the highest level of data privacy and integrity

SNMP Traps

Provides information on SNMP traps.

Description

The MPLAB Harmony TCP/IP Stack supports Trap version1 and Trap version2 formatted traps. Traps are notifications from the agent to the manager that are used when a predefined event occurs at the agent.

From `mchip.mib`, the ipv4TrapTable and ipv6TrapTable are the two tabular sections, which are used to configure HOST IPv4 and IPv6 address and this will help the HOST trap receiver to receive the traps.

Several preprocessor macros in the `snmp_config.h` variant header file can be used to enable or disable traps in the agent. Commenting and uncommenting these macros in the file will have different results. The SNMP_TRAP_DISABLED macro will disable traps entirely if it is not commented:

```
#define SNMP_TRAP_DISABLED
```

The user must configure the expected trap format at the SNMP Manager. SNMPv2 entities acting as an agent should be able to generate and transmit SNMP V2 trap PDUs when the manager is configured to receive and process SNMP V2 trap PDUs. To configure the trap format, comment or uncomment the SNMP_STACK_USE_V2_TRAP macro in the `snmp_config.h` header file:

```
#define SNMP_STACK_USE_V2_TRAP
```

If the macro has been commented out, the SNMP agent will send V1 formatted trap PDUs; otherwise, it will send V2 formatted trap PDUs. By default, the SNMP agent is configured to send V2 formatted traps. Note that the SNMP V2c agent should only send V2 formatted traps.

To enable traps in SNMPv3, the `#define SNMP_V1_V2_TRAP_WITH_SNMPv3` macro must be uncommented.

The following table illustrates how to enable/disable traps for different versions of SNMP.

Type	SNMPv1	SNMPv2c	SNMPv3
TRAPv2 (enabled by default)	Comment out the `#define SNMP_TRAP_DISABLED` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_V1_V2_TRAP_WITH_SNMPV3` macro
TRAPv2 (disabled by default)	Not supported	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_STACK_USE_V2_TRAP` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_V1_V2_TRAP_WITH_SNMPV3` macro Uncomment the `#define SNMP_STACK_USE_V2_TRAP` macro

Demonstrations

Two trap demonstrations APIs are included with the TCP/IP Stack. The task functions for these demonstrations are called in the main application function:

- SNMPTrapDemo - This API demonstrates V1 or V2 trap formats (depending of the status of the SNMP_STACK_USE_V2_TRAP macro). The trap PDU will only have one demonstration variable binding on the varbind list.
- SNMPV2TrapDemo - This API provides V2 format notifications with multiple (4) variable bindings. The user should modify or use this routine as a reference for sending V2 trap format notifications with multiple bindings on the varbind list.

Note: The user should only enable one SNMP demonstration API at a time. By default, the SNMPV2TrapDemo API is enabled and SNMPTrapDemo is commented out (disabled).

V1/V2 Formatted Traps with a Single Variable Binding

In the snmp_config.h header file:

- Uncomment `#define SNMP_TRAP_DISABLED`
- Comment `//#define SNMP_STACK_USE_V2_TRAP`

For the Trap demonstration, two events are defined within the V2c agent:

- If the Analog Potentiometer value is greater than 14, the agent will send a Trap every 5 seconds to the configured 'IPv4TrapReceiverIP address'.
- If Button 3 on the demonstration board is pressed, an organization-specific PUSH_BUTTON trap will be sent.

The current implementation of the V2c agent also generates a standard "Authentication Failure Trap":

- If a request is received to modify (Set) a private MIB variable, or
- If the value of the variable is requested (Get) by a browser with the wrong community name

HTTP Configuration

Provides information on the SNMP Community Configuration.

Description

If a HTTP2 server is used with the Microchip TCP/IP stack, it is possible to dynamically configure the Read and Write community names through the SNMP Configuration web page. Access the web page using http://mchpboard_e/mpfsupload or http://<Board IP address>(for IPv6 it should be http://<Ipv6 address>:80/index.html), and then access the SNMP Configuration web page through the navigation bar. Use "admin" for the username and "microchip" for the password.

Library Interface

a) SNMP Module Functions

	Name	Description
	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_TrapTimeGet	Gets SNMP Trap UDP client open socket time-out.
	TCPIP_SNMP_ClientGetNet	Get a network interface for SNMP TRAP.
	TCPIP_SNMP_ExactIndexGet	To search for exact index node in case of a Sequence variable.
	TCPIP_SNMP_IsValidCommunity	Validates community name for access control.
	TCPIP_SNMP_IsValidLength	Validates the set variable data length to data type.

	TCPIP_SNMP_MibIDSet	Sets the agent MIB ID for SNP notification.
	TCPIP_SNMP_NextIndexGet	To search for next index node in case of a Sequence variable.
	TCPIP_SNMP_ReadCommunityGet	Gets the readCommunity String with SNMP index.
	TCPIP_SNMP_RecordIDValidation	Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.
	TCPIP_SNMP_SendFailureTrap	Prepares and validates the remote node that will receive a trap and send the trap PDU.
	TCPIP_SNMP_TrapInterFaceSet	Sets the TRAP interface for SNMP notification.
	TCPIP_SNMP_TRAPMibIDGet	Gets the agent MIB ID for SNP notification.
	TCPIP_SNMP_TrapSendFlagGet	Gets the status of trap send flag.
	TCPIP_SNMP_TrapSendFlagSet	Sets the status of trap send flag.
	TCPIP_SNMP_TrapSpecificNotificationGet	Gets the specific trap.
	TCPIP_SNMP_TrapSpecificNotificationSet	Sets the specific trap, generic trap, and trap ID.
	TCPIP_SNMP_VarbindGet	Used to get/collect OID variable information.
	TCPIP_SNMP_VarbindSet	Sets the MIB variable with the requested value.
	TCPIP_SNMP_WriteCommunityGet	Gets the writeCommunity String with SNMP index.
	TCPIP_SNMP_AuthTrapFlagGet	Gets the status of authentication trap flag.
	TCPIP_SNMP_AuthTrapFlagSet	Sets the status of authentication trap flag.
	TCPIP_SNMP_IsTrapEnabled	Gets the SNMP Trap status.
	TCPIP_SNMP_TRAPTypeGet	Get SNMP Trap type for version v1 and v2c.
	TCPIP_SNMP_TRAPv1Notify	Creates and Sends TRAPv1 pdu.
	TCPIP_SNMP_TRAPv2Notify	Creates and sends TRAP PDU.
	TCPIP_SNMPV3_TrapTypeGet	Gets SNMP Trap type for version v3.
	TCPIP_SNMP_ValidateTrapIntf	Gets the status of trap interface.
	TCPIP_SNMP_ReadCommunitySet	Sets the readCommunity String with SNMP index.
	TCPIP_SNMP_WriteCommunitySet	Sets the writeCommunity String with SNMP index.
	TCPIP_SNMP_SocketIDGet	Gets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_SocketIDSet	Sets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_Task	Standard TCP/IP stack module task function.

b) SNMPv3 Module Functions

	Name	Description
	TCPIP_SNMPV3_EngineUserDataBaseGet	Get SNMPv3 engine data base details.
	TCPIP_SNMPV3_EngineUserDataBaseSet	Set SNMPv3 engine data base details.
	TCPIP_SNMPv3_TrapConfigDataGet	Gets the SNMPv3 Trap configuration details using the user name index.
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP PDU.

c) SNMP Data Types and Constants

	Name	Description
	SNMP_END_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.
	SNMP_INDEX_INVALID	This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.
	SNMP_V1	This macro is used for SNMP version 1
	SNMP_V2C	This macro is used for SNMP version 2 with community
	SNMP_V3	This macro is used for SNMP version 3 with authentication and privacy
	SNMP_COMMUNITY_TYPE	Definition to represent different type of SNMP communities.
	SNMP_ID	SNMP dynamic variable ID.
	SNMP_INDEX	SNMP sequence variable index.

	SNMP_NON_MIB_RECD_INFO	Restrict access for specific OIDs.
	SNMP_TRAP_IP_ADDRESS_TYPE	Definition of the supported address types for SNMP trap.
	SNMP_VAL	Definition to represent SNMP OID object values.
	SNMP_GENERIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP generic trap types.
	SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP vendor trap types.

d) SNMPv3 Data Types and Constants

	Name	Description
	SNMPV3_HMAC_HASH_TYPE	Different type of authentication for SNMPv3.
	SNMPV3_PRIV_PROT_TYPE	Different type of encryption and decryption for SNMPv3.
	STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL	Different SNMP Message processing model
	STD_BASED_SNMP_SECURITY_MODEL	Different Security services for SNMPv3 messages.
	STD_BASED_SNMPV3_SECURITY_LEVEL	Different Security Level for SNMPv3 messages.
	TCPIP_SNMPV3_USERDATABASECONFIG_TYPE	Different Configuration parameters of SNMPv3 operation
	TCPIP_SNMP_COMMUNITY_CONFIG	SNMP community configuration.
	TCPIP_SNMP_MODULE_CONFIG	SNMP module configuration.
	TCPIP_SNMPV3_TARGET_ENTRY_CONFIG	SNMP module trap target address configuration.
	TCPIP_SNMPV3_USM_USER_CONFIG	SNMPv3 USM configuration.

Description

This section describes the Application Programming Interface (API) functions of the SNMP module.

Refer to each section for a detailed description.

a) SNMP Module Functions

TCPIP_SNMP_NotifyIsReady Function

Resolves given remoteHost IP address into MAC address.

File

snmp.h

C

```
bool TCPIP_SNMP_NotifyIsReady(IP_MULTI_ADDRESS* remoteHost, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function resolves given remoteHost IP address into MAC address using ARP module. If remoteHost is not available, this function would never return true. Application must implement time out logic to handle "remoteHost not available" situation.

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyPrepare is already called.

Parameters

Parameters	Description
remoteHost	Pointer to remote Host IP address
eTrapMultiAddressType	IPv4 and IPv6 address type

Function

bool TCPIP_SNMP_NotifyIsReady(IP_MULTI_ADDRESS* remoteHost,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMP_NotifyPrepare Function

Collects trap notification info and send ARP to remote host.

File

snmp.h

C

```
void TCPIP_SNMP_NotifyPrepare(IP_MULTI_ADDRESS* remoteHost, char* community, uint8_t communityLen, SNMP_ID agentIDVar, uint8_t notificationCode, uint32_t timestamp);
```

Returns

None.

Description

This function prepares SNMP module to send SNMP trap notification to remote host. It sends ARP request to remote host to learn remote host MAC address.

Remarks

This is first of series of functions to complete SNMP notification.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Parameters

Parameters	Description
remoteHost	pointer to remote Host IP address
community	Community string to use to notify
communityLen	Community string length
agentIDVar	System ID to use identify this agent
notificaitonCode	Notification Code to use
timestamp	Notification timestamp in 100th of second.

Function

void TCPIP_SNMP_NotifyPrepare(IP_MULTI_ADDRESS* remoteHost,
char* community,
uint8_t communityLen,
SNMP_ID agentIDVar,
uint8_t notificationCode,
uint32_t timestamp)

TCPIP_SNMP_TrapTimeGet Function

Gets SNMP Trap UDP client open socket time-out.

File

snmp.h

C

```
uint32_t TCPIP_SNMP_TrapTimeGet();
```

Returns

uint32_t time

Description

This function returns a uint32_t time(snmpTrapTimer) which is used to time out a SNMP TRAP notification for a HOST. snmpTrapTimer is initialized when there is UDP client socket open either for a HOST IPv4 or IPv6 address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

uint32_t TCPIP_SNMP_TrapTimeGet(void)

TCPIP_SNMP_ClientGetNet Function

Get a network interface for SNMP TRAP.

File

snmp.h

C

```
TCPIP_NET_HANDLE TCPIP_SNMP_ClientGetNet(int * netIx, TCPIP_NET_HANDLE hNet);
```

Returns

TCPIP_NET_HANDLE

- Success - returns a valid interface
- Failure - no interface

Description

This function is used to get a network interface to transmit SNMP trap.

Remarks

None.

Preconditions

The SNMP module should be initialized.

Parameters

Parameters	Description
netIx	Network index
hNet	Network interface .If hNet is NULL, then a valid interface will returned.

Function

TCPIP_NET_HANDLE TCPIP_SNMP_ClientGetNet(int *netIx,TCPIP_NET_HANDLE hNet);

TCPIP_SNMP_ExactIndexGet Function

To search for exact index node in case of a Sequence variable.

File

snmp.h

C

```
bool TCPIP_SNMP_ExactIndexGet(SNMP_ID var, SNMP_INDEX * index);
```

Returns

-true - If the exact index value exists for a given variable at a given index. -false - If the exact index value does not exist for the given variable

Description

This is a callback function called by SNMP module. This function contents are modified by the application developer with the new MIB Record ID. This function can be called for individual MIB ID with index. This function is used to get the first available index from global or dynamic allocated sequence variable table.

Remarks

Only sequence index needs to be handled in this function and this function is called after TCPIP_SNMP_RecordIDValidation.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Parameters

Parameters	Description
var	Variable id as per mib.h (input)
index	Index of variable (input)

Function

bool TCPIP_SNMP_ExactIndexGet(SNMP_ID var,SNMP_INDEX *index)

TCPIP_SNMP_IsValidCommunity Function

Validates community name for access control.

File

snmp.h

C

```
uint8_t TCPIP_SNMP_IsValidCommunity(uint8_t* community);
```

Returns

This function returns the community validation result as READ_COMMUNITY or WRITE_COMMUNITY or INVALID_COMMUNITY.

Description

This function validates the community name for the MIB access to SNMP manager. The SNMP community name received in the request PDU is validated for read and write community names. The agent gives an access to the mib variables only if the community matches with the predefined values. This routine also sets a global flag to send trap if authentication failure occurs.

Remarks

This is a callback function called by module. User application must implement this function and verify that community matches with predefined value. This validation occurs for each SNMP manager request.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
community	Pointer to community string as sent by SNMP manager

Function

uint8_t TCPIP_SNMP_IsValidCommunity(uint8_t* community)

TCPIP_SNMP_IsValidLength Function

Validates the set variable data length to data type.

File

snmp.h

C

```
bool TCPIP_SNMP_IsValidLength(SNMP_ID var, uint8_t len, uint8_t index);
```

Returns

- true - if given var can be set to given len
- false - if the given var cannot be set to the given len

Description

This function is used to validate the dynamic variable data length to the variable data type. It is called before the SET request is processed. This is a callback function called by module. User application must implement this function.

Remarks

This function will be called for only dynamic variables that are defined as ASCII_STRING and OCTET_STRING.

Preconditions

TCPIP_SNMP_ProcessSetVar is called.

Parameters

Parameters	Description
var	Variable id whose value is to be set
len	Length value that is to be validated.

Function

bool TCPIP_SNMP_IsValidLength(SNMP_ID var, uint8_t len,uint8_t index)

TCPIP_SNMP_MibIDSet Function

Sets the agent MIB ID for SNP notification.

File

snmp.h

C

```
void TCPIP_SNMP_MibIDSet(uint32_t mibID);
```

Returns

None.

Description

This function is used to Set the MIB ID which is require while transmitting SNMP notification.SNMP user can this function without adding this to the Microchip MIB script.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
mibID	Trap client MIB ID

Function

void TCPIP_SNMP_MibIDSet(uint32_t *mibID)

TCPIP_SNMP_NextIndexGet Function

To search for next index node in case of a Sequence variable.

File

snmp.h

C

```
bool TCPIP_SNMP_NextIndexGet(SNMP_ID var, SNMP_INDEX* index);
```

Returns

- true - If a next index value exists for given variable at given index and index parameter contains next valid index.
- false - If a next index value does not exist for a given variable

Description

This is a callback function called by SNMP module. This function contents are modified by the application developer with the new MIB Record ID. This function can be called for individual MIB ID with index or for a complete MIB table without any instance.

- If the SNMP Sequence MIB variable processing is performed with index or instance value,

it is the responsibility of the agent to send the next available index for that requested variable. This is a only one iteration process. The Manager

will not send any further requests for the received index value.

- If SNMP request for a sequence variable starts with only OID without any instance, that is

a complete table is requested without any instance, it is the responsibility of the agent to send the first available index of the table. The Manager will continue the request with the transferred instance until it receives the reply from agent.

This function will only be called for OID variable of type sequence. if the index value starts with SNMP_INDEX_INVALID , then user need to send the response with first available index value.

Remarks

Only sequence index needs to be handled in this function and this function is called after TCPIP_SNMP_RecordIDValidation.

Preconditions

TCPIP_SNMP_ProcessVariables is called.

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Next Index of variable that should be transferred

Function

bool TCPIP_SNMP_NextIndexGet(SNMP_ID var,SNMP_INDEX* index)

TCPIP_SNMP_ReadCommunityGet Function

Gets the readCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_ReadCommunityGet(int index, int len, uint8_t * dest);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect READ community string from the global community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
dest	Copy the community string to this address and it should have a valid address.

Function

bool TCPIP_SNMP_ReadCommunityGet(int index,int len, uint8_t * dest)

TCPIP_SNMP_RecordIDValidation Function

Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.

File

snmp.h

C

```
bool TCPIP_SNMP_RecordIDValidation(uint8_t snmpVersion, bool idPresent, uint16_t varId, uint8_t *
oidValuePtr, uint8_t oidLen);
```

Returns

- true - A record ID exists
- false - A record ID does not exist

Description

This is a callback function called by SNMP module. SNMP user must implement this function as per SNMP version. One need to add the new SNMP MIB IDs here as per SNMP version (e.g., SYS_UP_TIME (250) is common for V1/V2/V3). ENGINE_ID - is the part of V3; therefore, place all of the SNMPv3 var IDs within the macro TCPIP_STACK_USE_SNMPV3_SERVER.

Remarks

This function is specific for record ID validation and this can also be used to restrict OID string.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
snmpVersion	different SNMP version
idPresent	true if SNMp record id is present else false
varId	dynamic record ID value as per mib.h
oidValuePtr	OID Value pointer
oidLen	number of OIDs present to be processed

Function

bool TCPIP_SNMP_RecordIDValidation(uint8_t snmpVersion,bool idPresent,

uint16_t varId,uint8_t * oidValuePtr,uint8_t oidLen)

TCPIP_SNMP_SendFailureTrap Function

Prepares and validates the remote node that will receive a trap and send the trap PDU.

File

snmp.h

C

```
void TCPIP_SNMP_SendFailureTrap();
```

Returns

None.

Description

This function is used to send trap notification to previously configured IP address if trap notification is enabled. There are different trap notification code. The current implementation sends trap for authentication failure (4).

Remarks

This is a callback function called by the application on certain predefined events. This routine only implemented to send a authentication failure Notification-type macro with PUSH_BUTTON OID stored in MPFS. If the ARP is not resolved (i.e., if TCPIP_SNMP_NotifyIsReady returns false, this routine times out in 5 seconds). This function should be modified according to event occurred and should update the corresponding OID and notification type to the TRAP PDU.

Preconditions

If the defined application event occurs to send the trap.

Function

void TCPIP_SNMP_SendFailureTrap(void)

TCPIP_SNMP_TrapInterFaceSet Function

Sets the TRAP interface for SNMP notification.

File

snmp.h

C

```
void TCPIP_SNMP_TrapInterFaceSet(TCPIP_NET_HANDLE netIntf);
```

Returns

None.

Description

This function is used to Set the network interface to which the TRAP socket is ready to transmit Notifications to the TRAP receiver address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
netIntf	Network interface Trap is connected to receiver

Function

void TCPIP_SNMP_TrapInterFaceSet(TCPIP_NET_HANDLE netIntf)

TCPIP_SNMP_TRAPMibIDGet Function

Gets the agent MIB ID for SNP notification.

File

snmp.h

C

```
void TCPIP_SNMP_TRAPMibIDGet(uint32_t * mibID);
```

Returns

None.

Description

This function is used to get the MIB ID which is require while transmitting SNMP notification.This MIB ID is a Agent ID value of the Microchip style MIB script. MIB ID macro value is present in mib.h which is generated by mib2bib.jar.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
mibID	Trap client MIB ID

Function

void TCPIP_SNMP_TRAPMibIDGet(uint32_t *mibID)

TCPIP_SNMP_TrapSendFlagGet Function

Gets the status of trap send flag.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSendFlagGet(bool * trapNotify);
```

Returns

None.

Description

This function is used to get the trap send flag details and this is used only when user is trying to send more than one varbind in a single PDU. That is more than one notification details are required to be part of a single PDU.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

void TCPIP_SNMP_TrapSendFlagGet(bool *trapNotify)

TCPIP_SNMP_TrapSendFlagSet Function

Sets the status of trap send flag.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSendFlagSet(bool trapNotify);
```

Returns

None.

Description

This function is used to set the trap send flag details and this is used only when user is trying to send more than one varbind in a single PDU. That is more than one notification details are required to be part of a single PDU. By default TRAP send flag is set to false. If there is only varbind need to be part of Notification PDU, then this function should be called with boolean false. Please find the usage of this flag in this following example.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
void SNMPv2TrapDemo(void)
{
    //set TRAP send flag to true , it signifies that there are more than one
    // variable need to be the part of SNMP v2 TRAP.
    TCPIP_SNMP_TrapSendFlagSet(true);

    // PUSH button varbind
    TCPIP_SNMP_Notify(PUSH_BUTTON,analogPotVal,0);

    // Before adding the last varbind to the TRAP PDU, TRAP send flag should
    // be set to  false. That it indicates it is the last varbind to the
    // Notification PDU.
```

```
    TCPIP_SNMP_TrapSendFlagSet(false);

    // Last varbind LED0_IO
    TCPIP_SNMP_Notify(LED0_IO,analogPotVal,0);

}
```

Function

void TCPIP_SNMP_TrapSendFlagSet(bool trapNotify)

TCPIP_SNMP_TrapSpecificNotificationGet Function

Gets the specific trap.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSpecificNotificationGet(uint8_t * specTrap);
```

Returns

None.

Description

This function is used to get specific trap value. Specific trap values are listed in SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
specTrap	Vendor specific trap value

Function

void TCPIP_SNMP_TrapSpecificNotificationGet(uint8_t *specTrap)

TCPIP_SNMP_TrapSpecificNotificationSet Function

Sets the specific trap, generic trap, and trap ID.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSpecificNotificationSet(uint8_t specTrap, uint8_t genTrap, SNMP_ID trapID);
```

Returns

None.

Description

The SNMP user needs to call this function before transmitting any traps. This function will set the vendor specific trap, generic trap, and default trap ID value. The SNMPv2 trap will use this trap ID while sending a specific trap.

Remarks

The Trap ID is the NOTIFICATION-TYPE of the ASN.1 MIB format. From the Trap ID, the SNMP agent will be able to obtain the OID string, which will be used while preparing TRAPv2 second varbind.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
specTrap	Vendor specific trap value (enumeration value of SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE)
genTrap	Generic trap (enumeration value of SNMP_GENERIC_TRAP_NOTIFICATION_TYPE)
trapID	Trap ID

Function

void TCPIP_SNMP_TrapSpecificNotificationSet(uint8_t specTrap,uint8_t genTrap,
SNMP_ID trapID)

TCPIP_SNMP_VarbindGet Function

Used to get/collect OID variable information.

File

snmp.h

C

```
bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index, uint8_t* ref, SNMP_VAL* val);
```

Returns

- true - If a value exists for given variable
- false - If a value does not exist for a given variable

Description

This is a callback function called by SNMP module. This function is called only when SNMP GET, GETNEXT and GETBULK request is made. This function content is modified by the application developer. This function contains the implementation of READ enable MIB OID macros.

mib2bib.jar Microchip MIB compiler utility is used to generate mib.h, which lists all the MIB OID ID value.

Remarks

This function may get called more than once depending on number of bytes in a specific get request for given variable. only dynamic read variables needs to be handled.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Example

Usage of MIB OID within this function for a ASCII string (multi byte variable)

```
// In the beginning *ref is equal to SNMP_START_OF_VAR
myRef = *ref;
switch(var)
{
// TRAP_COMMUNITY - generated from the Microchip style MIB script using mib2bib.jar,
// is a Sequence variable.
    case TRAP_COMMUNITY:
        if ( index < trapInfo.Size )
        {
        // check if the myRef should not cross the maximum length
             if ( myRef == trapInfo.table[index].communityLen )
             {
             // End of SNMP GET process
                 *ref = SNMP_END_OF_VAR;
                 return true;
             }
            if ( trapInfo.table[index].communityLen == 0u )
                *ref = SNMP_END_OF_VAR; // End of SNMP GET process
            else
            {
                // Start of SNMP GET process byte by byte
                val->byte = trapInfo.table[index].community[myRef];
                myRef++;
```

```
                *ref = myRef;
            }
            return true;
        }
        break;
// LED_D5 - generated from the Microchip style MIB script using mib2bib.jar is a scalar variable
    case LED_D5:
        val->byte = LED2_IO;
        return true;
}
```

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	For a scalar variable , Index value is zero. For sequence variable index value specifies which index value need to be set.
ref	Variable reference used to transfer multi-byte data. It is always SNMP_START_OF_VAR when very first byte is requested. Otherwise, use this as a reference to keep track of multi-byte transfers.
val	Pointer to up to 4 byte buffer: • If var data type is uint8_t, transfer data in val->byte • If var data type is uint16_t, transfer data in val->word • If var data type is uint32_t, transfer data in val->dword • If var data type is IP_ADDRESS, transfer data in val->v[] or val->dword • If var data type is COUNTER32, TIME_TICKS or GAUGE32, transfer data in val->dword • If var data type is ASCII_STRING or OCTET_STRING transfer data in val->byte using multi-byte transfer mechanism.

Function

bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index,uint8_t* ref, SNMP_VAL* val)

TCPIP_SNMP_VarbindSet Function

Sets the MIB variable with the requested value.

File

snmp.h

C

```
bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index, uint8_t ref, SNMP_VAL val);
```

Returns

- true - if it is okay to set more byte(s)
- false - if it is not okay to set more byte(s)

Description

This is a callback function called by module for SNMP SET request. This function content is modified by the application developer. This function contains the implementation of WRITE enable MIB OID macros.

mib2bib.jar Microchip MIB compiler utility is used to generate mib.h, which lists all the MIB OID ID value.

Remarks

This function may get called more than once depending on number of bytes in a specific set request for given variable. only dynamic read/-write variables needs to be handled.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Example

```
switch(var)
{
    // LED_D5 - generated from the Microchip style MIB script using mib2bib.jar is a scalar variable
    case LED_D5:
```

```
        LED2_IO = val.byte;
        return true;

    // TRAP_COMMUNITY - generated from the Microchip style MIB script using mib2bib.jar,
    // is a Sequence variable.

    case TRAP_COMMUNITY:
        // Since this is a ASCII_STRING data type, SNMP will call with
        // SNMP_END_OF_VAR to indicate no more bytes or end of SNMP SET process
        // Use this information to determine if we just added new row
        // or updated an existing one.
        if ( ref ==  SNMP_END_OF_VAR )
        {
            // Index equal to table size means that we have new row.
            if ( index == trapInfo.Size )
                trapInfo.Size++;

            // Length of string is one more than index.
            trapInfo.table[index].communityLen++;

            return true;
        }

        // Make sure that index is within our range.
        if ( index < trapInfo.Size )
        {
            // Copy given value into local buffer.
            trapInfo.table[index].community[ref] = val.byte;
            // Keep track of length too.
            // This may not be NULL terminate string.
            trapInfo.table[index].communityLen = (uint8_t)ref;
            return true;
        }
        break;
}
```

Parameters

Parameters	Description
var	Write enable Variable id whose value is to be set
index	For a scalar variable , Index value is zero. For sequence variable index value specifies which index value need to be set.
ref	Variable reference used to transfer multi-byte data 0 if first byte is set otherwise non zero value to indicate corresponding byte being set. SNMP set will performed until ref is not equal to SNMP_END_OF_VAR and SNMP set starts with ref = SNMP_START_OF_VAR.
val	Up to 4 byte data value: • If var data type is uint8_t, variable value is in val->byte • If var data type is uint16_t, variable value is in val->word • If var data type is uint32_t, variable value is in val->dword. • If var data type is IP_ADDRESS, COUNTER32, or GAUGE32, value is in val->dword • If var data type is OCTET_STRING, ASCII_STRING value is in val->byte; multi-byte transfer will be performed to transfer remaining bytes of data.

Function

bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index,
uint8_t ref, SNMP_VAL val)

TCPIP_SNMP_WriteCommunityGet Function

Gets the writeCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_WriteCommunityGet(int index, int len, uint8_t * dest);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect WRITE community string from the global community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
dest	Copy the community string to this address and it should have a valid address.

Function

bool TCPIP_SNMP_WriteCommunityGet(int index,int len, uint8_t * dest)

TCPIP_SNMP_AuthTrapFlagGet Function

Gets the status of authentication trap flag.

File

snmp.h

C

```
void TCPIP_SNMP_AuthTrapFlagGet(bool * authTrap);
```

Returns

None.

Description

This function is used to Get the authentication trap send flag and this is used only when the user is trying to send authentication failure trap. For example, sending a trap if community does not match the global community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
sendTrap	Trap flag value.

Function

void TCPIP_SNMP_AuthTrapFlagGet(bool *sendTrap)

TCPIP_SNMP_AuthTrapFlagSet Function

Sets the status of authentication trap flag.

File

snmp.h

C

```
void TCPIP_SNMP_AuthTrapFlagSet(bool sendTrap);
```

Returns

None.

Description

This function is used to set the authentication trap send flag and this is used only when user is trying to send authentication failure trap. Ex-sending a trap if community do not match to the global community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

void TCPIP_SNMP_AuthTrapFlagSet(bool sendTrap)

TCPIP_SNMP_IsTrapEnabled Function

Gets the SNMP Trap status.

File

snmp.h

C

```
bool TCPIP_SNMP_IsTrapEnabled();
```

Returns

- true - SNMP module has trap enabled
- false - SNMP module has trap disabled

Description

This function returns true if SNMP module is enabled to send the trap to to the SNMP manager device.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

bool TCPIP_SNMP_IsTrapEnabled(void)

TCPIP_SNMP_TRAPTypeGet Function

Get SNMP Trap type for version v1 and v2c.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPTypeGet();
```

Returns

- true - trap version is v2
- false - trap version is v1

Description

This function returns true if the trap tye is v2 and the TRAP pdu packet will be a TRAP v2 format.The return value is also used is validated when

SNMP module is trying to send a trap for SNMP version v3.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

bool TCPIP_SNMP_TRAPTypeGet(void)

TCPIP_SNMP_TRAPv1Notify Function

Creates and Sends TRAPv1 pdu.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPv1Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function creates SNMP trap PDU and sends it to previously specified remoteHost. snmpv1 trap pdu: | PDU-type | enterprise | agent-addr | generic-trap | specific-trap | | time-stamp | varbind-list |

The v1 enterprise is mapped directly to SNMPv2TrapOID.0

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyIsReady() is already called and returned true.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127
eTrapMultiAddressType	Trap address type

Function

bool TCPIP_SNMP_TRAPv1Notify(SNMP_ID var, SNMP_VAL val,
SNMP_INDEX index,SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMP_TRAPv2Notify Function

Creates and sends TRAP PDU.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPv2Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function creates SNMP V2 Trap PDU and sends it to previously specified remoteHost.

SNMP V1 trap PDU: | PDU type | enterprise | agent addr | generic trap | specific trap | | time stamp | varbind list |

The v1 enterprise is mapped directly to SNMPv2TrapOID.0

SNMP V2 trap PDU: version (0 or 1) | community | SNMP PDU |PDU type | request id | error status |err index |varbinds

The first two variables (in varbind list) of snmpv2 are: sysUpTime.0 and SNMPv2TrapOID.0

Generic Trap OID is used as the varbind for authentication failure.

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyIsReady is already called and returned true.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127
eTrapMultiAddressType	Trap address type

Function

bool TCPIP_SNMP_TRAPv2Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMPV3_TrapTypeGet Function

Gets SNMP Trap type for version v3.

File

snmp.h

C

```
bool TCPIP_SNMPV3_TrapTypeGet();
```

Returns

- true - SNMP3 will use trap version v2
- false - SNMP3 will use trap version v1

Description

This function returns true if SNMP module is trying to send trap version v2 with SNMP version v3.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

bool TCPIP_SNMPV3_TrapTypeGet(void)

TCPIP_SNMP_ValidateTrapIntf Function

Gets the status of trap interface.

File

snmp.h

C

```
bool TCPIP_SNMP_ValidateTrapIntf(TCPIP_NET_HANDLE pIf);
```

Returns

- true - SNMP trap interface is valid
- false - SNMP trap interface is invalid

Description

This function returns true if SNMP module trap interface is a valid interface.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
pif	network interface

Function

bool TCPIP_SNMP_ValidateTrapIntf(TCPIP_NET_HANDLE pIf)

TCPIP_SNMP_ReadCommunitySet Function

Sets the readCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_ReadCommunitySet(int index, int len, uint8_t * src);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to configure READ community string from the user and configure the SNMP community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
src	Copy this community string to snmp community table.

Function

bool TCPIP_SNMP_ReadCommunitySet(int index,int len, uint8_t * src)

TCPIP_SNMP_WriteCommunitySet Function

Sets the writeCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_WriteCommunitySet(int index, int len, uint8_t * src);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect WRITE community string from user and set the community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
src	Copy this community string to SNMP community table.

Function

bool TCPIP_SNMP_WriteCommunitySet(int index,int len, uint8_t * src)

TCPIP_SNMP_SocketIDGet Function

Gets the Socket ID for SNMP Server socket.

File

snmp.h

C

```
void TCPIP_SNMP_SocketIDGet(UDP_SOCKET * socket);
```

Returns

None.

Description

This function is used to get trap client socket ID for both IPv4 and IPv6 receiver Address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
socket	Trap client socket ID

Function

void TCPIP_SNMP_SocketIDGet(UDP_SOCKET *socket)

TCPIP_SNMP_SocketIDSet Function

Sets the Socket ID for SNMP Server socket.

File

snmp.h

C

```
void TCPIP_SNMP_SocketIDSet(UDP_SOCKET socket);
```

Returns

None.

Description

This function is used to update socket value of SNMP trap global structure.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
socket	Trap client socket ID

Function

void TCPIP_SNMP_SocketIDSet(UDP_SOCKET socket)

TCPIP_SNMP_Task Function

Standard TCP/IP stack module task function.

File

snmp.h

C

```
void TCPIP_SNMP_Task();
```

Returns

None.

Description

This function performs SNMP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The SNMP module should have been initialized

Function

void TCPIP_SNMP_Task(void)

b) SNMPv3 Module Functions

TCPIP_SNMPV3_EngineUserDataBaseGet Function

Get SNMPv3 engine data base details.

File

snmpv3.h

C

```
bool TCPIP_SNMPV3_EngineUserDataBaseGet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType, uint8_t len,
uint8_t userIndex, void * val);
```

Returns

None.

Description

This function is used to get SNMPv3 Engine data base details using TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration.

Remarks

None

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index, uint8_t* ref, SNMP_VAL* val)
{
    switch(var)
    {
        case USER_SECURITY_NAME:
            if(index < TCPIP_SNMPV3_USM_MAX_USER)
            {
                if(TCPIP_SNMPV3_EngineUserDataBaseGet(SNMPV3_USERNAME_CONFIG_TYPE,myRef,index,&val->byte)
== false)
                {
                    *ref = SNMP_END_OF_VAR;
                }
                else
                {
                    myRef++;
                    *ref = myRef;
                }
                return true;
            }
            break;
    }
}
```

Parameters

Parameters	Description
userDataBaseType	SNMPv3 data base configuration type
len	Number of bytes need to be read from data base
userIndex	SNMPv3 user index
val	void pointer to a any data type. Length parameter value is changed according to data type.

Function

void TCPIP_SNMPV3_EngineUserDataBaseGet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType,
uint8_t len,uint8_t userIndex,void *val);

TCPIP_SNMPV3_EngineUserDataBaseSet Function

Set SNMPv3 engine data base details.

File

snmpv3.h

C

```
bool TCPIP_SNMPV3_EngineUserDataBaseSet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType, uint8_t len,
uint8_t userIndex, void * val);
```

Returns

None.

Description

This function is used to set SNMPv3 Engine data base details using TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index, uint8_t ref, SNMP_VAL val)
{
  switch(var)
  {
      case USER_SECURITY_NAME:
          if ( ref ==  SNMP_END_OF_VAR )
          {
              if(TCPIP_SNMPV3_EngineUserDataBaseSet(SNMPV3_USERNAME_CONFIG_TYPE,strlen((char*)gSnmpv3UserSec
urityName),
                                                  index,gSnmpv3UserSecurityName) != true)
                  return false;
          }
          // Make sure that index is within our range.
          if ( index < TCPIP_SNMPV3_USM_MAX_USER )
          {
              // Copy given value into local buffer.
              gSnmpv3UserSecurityName[ref]=val.byte;
              return true;
          }
          break;
  }
}
```

Parameters

Parameters	Description
userDataBaseType	SNMPv3 data base configuration type
len	Number of bytes need to be read from data base
userIndex	SNMPv3 user index
val	void pointer to a any data type. Length parameter value is changed according to data type.

Function

void TCPIP_SNMPV3_EngineUserDataBaseSet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType, uint8_t len,uint8_t userIndex,void *val);

TCPIP_SNMPv3_TrapConfigDataGet Function

Gets the SNMPv3 Trap configuration details using the user name index.

File

snmpv3.h

C

```
void TCPIP_SNMPv3_TrapConfigDataGet(uint8_t userIndex, uint8_t * msgModelType, uint8_t * securityModelType);
```

Returns

None.

Description

This function is used to get SNMPv3 message model type and security model type using user index.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
userIndex	user name index
msgModelType	message processing type
securityModelType	security model type

Function

void TCPIP_SNMPv3_TrapConfigDataGet(uint8_t userIndex,uint8_t *msgModelType,
uint8_t *securityModelType)

TCPIP_SNMPv3_Notify Function

Creates and Sends SNMPv3 TRAP PDU.

File

snmpv3.h

C

```
bool TCPIP_SNMPv3_Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, uint8_t targetIndex,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType);
```

Description

This function creates SNMPv3 trap PDU and sends it to previously specified remoteHost.

Remarks

None.

Preconditions

TRAP event is triggered.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127 targetIndex -index of the 'Snmpv3TrapConfigData' table's security user name for which the TRAP PDU message header to constructed.

Function

bool TCPIP_SNMPv3_Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index,
uint8_t targetIndex)

c) SNMP Data Types and Constants

SNMP_END_OF_VAR Macro

File

snmp.h

C

```
#define SNMP_END_OF_VAR (0xff)
```

Description

This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.

SNMP_INDEX_INVALID Macro

File

snmp.h

C

```
#define SNMP_INDEX_INVALID (0xff)
```

Description

This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.

SNMP_START_OF_VAR Macro

File

snmp.h

C

```
#define SNMP_START_OF_VAR (0)
```

Description

This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.

SNMP_V1 Macro

File

snmp.h

C

```
#define SNMP_V1 (0)
```

Description

This macro is used for SNMP version 1

SNMP_V2C Macro

File

snmp.h

C

```
#define SNMP_V2C (1)
```

Description

This macro is used for SNMP version 2 with community

SNMP_V3 Macro

File

snmp.h

C

```
#define SNMP_V3 (3)
```

Description

This macro is used for SNMP version 3 with authentication and privacy

SNMP_COMMUNITY_TYPE Enumeration

Definition to represent different type of SNMP communities.

File

snmp.h

C

```
typedef enum {
  READ_COMMUNITY = 1,
  WRITE_COMMUNITY = 2,
  INVALID_COMMUNITY = 3
} SNMP_COMMUNITY_TYPE;
```

Members

Members	Description
READ_COMMUNITY = 1	Read only community
WRITE_COMMUNITY = 2	Read write community
INVALID_COMMUNITY = 3	Community invalid

Description

Enumeration: SNMP_COMMUNITY_TYPE

List of different SNMP community types.

Remarks

SNMP agent use these community types for both TRAP and SNMP agent request.

SNMP_ID Type

SNMP dynamic variable ID.

File

snmp.h

C

```
typedef uint32_t SNMP_ID;
```

Description

Type: SNMP_ID

Only dynamic and AgentID variables can contain a dyanmic ID. MIB2BIB utility enforces this rule when BIB was generated. All the dynamic IDs are are listed in mib.h. The maximum value of a dynamic ID is 1023.

Remarks

mib2bib.jar utility generates mib.h and snmp.bib from Microchip MIB script. DynamicVar - This command declares defined OID variable as dynamic. Syntax - $DynamicVar(oidName, id).

SNMP_INDEX Type

SNMP sequence variable index.

File

snmp.h

C

```
typedef uint8_t SNMP_INDEX;
```

Description

Type: SNMP_INDEX

The current version limits the size of the index to 7 bits wide, meaning that such arrays can contain up to 127 entries.

Remarks

SequenceVar - This command is part of Microchip MIB script declares a previously defined OID variable as a sequence variable and assigns an index to it. A sequence variable can contain an array of values and any instance of its values can be referenced by an index. More than one sequence variable may share a single index, creating multi dimensional arrays.

More than one index variable is not supported by mib2bib.jar compiler.

SNMP_NON_MIB_RECD_INFO Structure

Restrict access for specific OIDs.

File

snmp.h

C

```
typedef struct {
  uint8_t oidstr[16];
  uint8_t version;
} SNMP_NON_MIB_RECD_INFO;
```

Description

Structure: SNMP_NON_MIB_RECD_INFO

This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version.OID string length is restricted to 16.

Remarks

None.

SNMP_TRAP_IP_ADDRESS_TYPE Enumeration

Definition of the supported address types for SNMP trap.

File

snmp.h

C

```
typedef enum {
  IPV4_SNMP_TRAP = 1,
  IPV6_SNMP_TRAP
} SNMP_TRAP_IP_ADDRESS_TYPE;
```

Description

Enumeration: SNMP_TRAP_IP_ADDRESS_TYPE

SNMP agent supports both IPv4 and IPv6 trap address type and is able to transmit traps to both IPv4 and IPv6 Host receiver address.

Remarks

None.

SNMP_VAL Union

Definition to represent SNMP OID object values.

File

snmp.h

C

```
typedef union {
  uint32_t dword;
  uint16_t word;
  uint8_t byte;
  uint8_t v[sizeof(uint32_t)];
} SNMP_VAL;
```

Members

Members	Description
uint32_t dword;	double word value
uint16_t word;	word value
uint8_t byte;	byte value
uint8_t v[sizeof(uint32_t)];	byte array

Description

Union: SNMP_VAL

SNMP agent processes different variable types.

Remarks

None.

SNMP_GENERIC_TRAP_NOTIFICATION_TYPE Enumeration

Definition to represent different SNMP generic trap types.

File

snmp.h

C

```
typedef enum {
  COLD_START = 0x0,
  WARM_START = 0x1,
  LINK_DOWN = 0x2,
  LINK_UP = 0x3,
  AUTH_FAILURE = 0x4,
  EGP_NEIGHBOR_LOSS = 0x5,
  ENTERPRISE_SPECIFIC = 0x6
} SNMP_GENERIC_TRAP_NOTIFICATION_TYPE;
```

Members

Members	Description
COLD_START = 0x0	Controls the sending of SNMP coldstart notifications. A coldStart(0) trap signifies that the sending protocol entity is reinitializing itself such that the configuration of the agent or the protocol entity implementation might be altered.
WARM_START = 0x1	Controls the sending of SNMP warmstart notifications. A warmStart(1) trap signifies that the sending protocol entity is reinitializing itself so that neither the agent configuration nor the protocol entity implementation can be altered.
LINK_DOWN = 0x2	Controls the how SNMP linkdown notifications are sent. A linkDown(2) trap signifies that the sending protocol entity recognizes a failure in one of the communication links represented in the configuration of the agent.
LINK_UP = 0x3	Controls the sending of SNMP linkup notifications. A linkUp(3) trap signifies that the sending protocol entity recognizes that one of the communication links represented in the configuration of the agent has come up.
AUTH_FAILURE = 0x4	Controls the distribution of SNMP authentication failure notifications. An authenticationFailure(4) trap signifies that the sending protocol entity is the addressee of a protocol message that is not properly authenticated. Like Community Name authentication failure
EGP_NEIGHBOR_LOSS = 0x5	Controls the distribution of SNMP egpNeighborLoss notifications.An egpNeighborLoss(5) trap signifies that an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer exists.
ENTERPRISE_SPECIFIC = 0x6	Controls the distribution of SNMP enterprise/-specific notifications.An enterpriseSpecific(6) trap signifies that the sending protocol entity recognizes that some enterprise/-specific event has occurred. The specific-trap field identifies the particular trap that occurred.

Description

Enumeration: SNMP_GENERIC_TRAP_NOTIFICATION_TYPE

List of different SNMP specific Notification types.

Remarks

ENTERPRISE_SPECIFIC and AUTH_FAILURE are used while sending specific trap.

SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration

Definition to represent different SNMP vendor trap types.

File

snmp.h

C

```
typedef enum {
  VENDOR_TRAP_DEFAULT = 0x0,
  BUTTON_PUSH_EVENT = 0x1,
  POT_READING_MORE_512 = 0x2
} SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE;
```

Members

Members	Description
VENDOR_TRAP_DEFAULT = 0x0	Default trap . Agent send use this trap when there is authentication failure.
BUTTON_PUSH_EVENT = 0x1	PUSH button event trap notification
POT_READING_MORE_512 = 0x2	Analog POT meter event trap notification

Description

Enumeration: SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE

List of different SNMP Vendor Notification types.

Remarks

None.

d) SNMPv3 Data Types and Constants

SNMPV3_HMAC_HASH_TYPE Enumeration

Different type of authentication for SNMPv3.

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_HMAC_MD5 = 0u,
  SNMPV3_HMAC_SHA1,
  SNMPV3_NO_HMAC_AUTH
} SNMPV3_HMAC_HASH_TYPE;
```

Members

Members	Description
SNMPV3_HMAC_MD5 = 0u	MD5 is being calculated HMAC SHA1 authentication protocol
SNMPV3_HMAC_SHA1	SHA-1 is being calculated No authentication is supported

Description

Enumeration: SNMPV3_HMAC_HASH_TYPE

The following authentication types are supported by the SNMPv3 USM model for data confidentiality. SNMPv3 agent supports both MD5 and SHA1 protocol for authentication.

Remarks

None.

SNMPV3_PRIV_PROT_TYPE Enumeration

Different type of encryption and decryption for SNMPv3.

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_DES_PRIV = 0x0,
  SNMPV3_AES_PRIV,
  SNMPV3_NO_PRIV
} SNMPV3_PRIV_PROT_TYPE;
```

Members

Members	Description
SNMPV3_DES_PRIV = 0x0	Data Encryption Standard (DES-CBC) encryption and decryption protocol
SNMPV3_AES_PRIV	Advanced Encryption Standard (AES-CFB) encryption and decryption protocol
SNMPV3_NO_PRIV	No encryption or decryption protocol is supported

Description

Enumeration: SNMPV3_PRIV_PROT_TYPE

These below privacy types are supported by the SNMPv3 USM model for data confidentiality. SNMPv3 agent supports both AES-CFB and DES-CBC encryption and decryption algorithm. For DES-CBC privacy protocol, SNMPv3 agent will use Harmony Crypto Library. For AES-CFB privacy protocol, SNMPv3 agent will use Legacy TCP/IP Crypto Library (For AES, include -aes_pic32mx.a to the project.)

Remarks

128-bit, 192-bit and 256-bit AES are supported.

STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL Enumeration

Different SNMP Message processing model

File

snmpv3.h

C

```
typedef enum {
  SNMPV1_MSG_PROCESSING_MODEL = 0X00,
  SNMPV2C_MSG_PROCESSING_MODEL = 0X01,
  SNMPV2U_V2_MSG_PROCESSING_MODEL = 0X02,
  SNMPV3_MSG_PROCESSING_MODEL = 0X03
} STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL;
```

Members

Members	Description
SNMPV1_MSG_PROCESSING_MODEL = 0X00	SNMP version 1 Message processing model
SNMPV2C_MSG_PROCESSING_MODEL = 0X01	SNMP version 2 Message processing model with community as security feature
SNMPV2U_V2_MSG_PROCESSING_MODEL = 0X02	SNMP version 2 Message processing model
SNMPV3_MSG_PROCESSING_MODEL = 0X03	SNMP version 3 Message processing model with authentication and encryption and decryption

Description

Enumeration: STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL

SNMP Message processing model is responsible for processing an SNMP version specific message and for coordinating the interaction with security sub system. SNMP message processing subsystem is part of an SNMP engine which interacts with the Dispatcher to handle the version specific SNMP messages.

A Message Processing Model describes the version-specific procedures for extracting data from messages, generating messages, calling upon a securityModel to apply its security services to messages.

Remarks

None.

STD_BASED_SNMP_SECURITY_MODEL Enumeration

Different Security services for SNMPv3 messages.

File

snmpv3.h

C

```
typedef enum {
  ANY_SECUTIRY_MODEL = 0x00,
  SNMPV1_SECURITY_MODEL = 0X01,
  SNMPV2C_SECURITY_MODEL = 0X02,
  SNMPV3_USM_SECURITY_MODEL = 0X03
} STD_BASED_SNMP_SECURITY_MODEL;
```

Members

Members	Description
ANY_SECUTIRY_MODEL = 0x00	Security Model Reserved for ANY
SNMPV1_SECURITY_MODEL = 0X01	Security Model reserved fro SNMP version 1
SNMPV2C_SECURITY_MODEL = 0X02	Community Security Model reserved for SNMP version 2
SNMPV3_USM_SECURITY_MODEL = 0X03	User based security model reserved for SNMP version 3

Description

Enumeration: STD_BASED_SNMP_SECURITY_MODEL

SNMP Security subsystem is applied to the transmission and reception of messages and to the processing of the contents of messages.

- The zero value does not identify any particular security model.
- Values between 1 and 255, inclusive, are reserved for standards-track Security Models and are

managed by the Internet Assigned Numbers Authority (IANA).

- Values greater than 255 are allocated to enterprise-specific Security Models. An

enterprise specific securityModel value is defined to be: enterpriseID * 256 + security model within enterprise

Remarks

None.

STD_BASED_SNMPV3_SECURITY_LEVEL Enumeration

Different Security Level for SNMPv3 messages.

File

snmpv3.h

C

```
typedef enum {
  NO_AUTH_NO_PRIV = 1,
  AUTH_NO_PRIV,
  AUTH_PRIV
} STD_BASED_SNMPV3_SECURITY_LEVEL;
```

Members

Members	Description
NO_AUTH_NO_PRIV = 1	without authentication and without privacy
AUTH_NO_PRIV	with authentication but without privacy
AUTH_PRIV	with authentication but with privacy

Description

Enumeration: STD_BASED_SNMPV3_SECURITY_LEVEL

A Level of Security at which SNMPv3 messages can be sent or with which operations are being processed.

Remarks

None.

TCPIP_SNMPV3_USERDATABASECONFIG_TYPE Enumeration

Different Configuration parameters of SNMPv3 operation

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_USERNAME_CONFIG_TYPE = 0,
  SNMPV3_AUTHPASSWD_CONFIG_TYPE,
  SNMPV3_PRIVPASSWD_CONFIG_TYPE,
  SNMPV3_AUTHPASSWDLOCALIZEDKEY_CONFIG_TYPE,
  SNMPV3_PRIVPASSWWDLOCALIZEDKEY_CONFIG_TYPE,
  SNMPV3_HASHTYPE_CONFIG_TYPE,
  SNMPV3_PRIVTYPE_CONFIG_TYPE,
  SNMPV3_TARGET_SECURITY_NAME_TYPE,
  SNMPV3_TARGET_SECURITY_LEVEL_TYPE,
  SNMPV3_TARGET_SECURITY_MODEL_TYPE,
  SNMPV3_TARGET_MP_MODEL_TYPE,
  SNMPV3_ENGINE_ID_TYPE,
  SNMPV3_ENGINE_BOOT_TYPE,
  SNMPV3_ENGINE_TIME_TYPE,
  SNMPV3_ENGINE_MAX_MSG_TYPE
} TCPIP_SNMPV3_USERDATABASECONFIG_TYPE;
```

Members

Members	Description
SNMPV3_USERNAME_CONFIG_TYPE = 0	Snmpv3 user name configuration
SNMPV3_AUTHPASSWD_CONFIG_TYPE	Authentication configuration type
SNMPV3_PRIVPASSWD_CONFIG_TYPE	Encryption and Decryption password configuration
SNMPV3_AUTHPASSWDLOCALIZEDKEY_CONFIG_TYPE	Authorization localized key configuration type
SNMPV3_PRIVPASSWWDLOCALIZEDKEY_CONFIG_TYPE	Encryption and Decryption password localized key configuration type
SNMPV3_HASHTYPE_CONFIG_TYPE	SNMPv3 hash algorithm type
SNMPV3_PRIVTYPE_CONFIG_TYPE	SNMPv3 privacy configuration type
SNMPV3_TARGET_SECURITY_NAME_TYPE	SNMPv3 target address user name . This is for Trap communication
SNMPV3_TARGET_SECURITY_LEVEL_TYPE	SNMPv3 target security type
SNMPV3_TARGET_SECURITY_MODEL_TYPE	SNMPv3 target security model type
SNMPV3_TARGET_MP_MODEL_TYPE	SNMPv3 target security message processing model type
SNMPV3_ENGINE_ID_TYPE	Identifier that uniquely and unambiguously identifies the local SNMPv3 engine
SNMPV3_ENGINE_BOOT_TYPE	Number of times the local SNMPv3 engine has rebooted or reinitialized since the engine ID was last changed
SNMPV3_ENGINE_TIME_TYPE	Number of seconds since the local SNMPv3 engine was last rebooted or reinitialized
SNMPV3_ENGINE_MAX_MSG_TYPE	SNMPv3 Engine Maximum message size the sender can accommodate

Description

Enumeration: TCPIP_SNMPV3_USERDATABASECONFIG_TYPE

These configuration types are used by the SNMP user while doing configuration SNMPv3 parameters. It includes SNMpv3 user name , authentication and encryption configuration parameters.

Remarks

None.

TCPIP_SNMP_COMMUNITY_CONFIG Structure

SNMP community configuration.

File

snmpv3.h

C

```
typedef struct {
  char * communityName;
} TCPIP_SNMP_COMMUNITY_CONFIG;
```

Description

Structure: TCPIP_SNMP_COMMUNITY_CONFIG

This structure is used to configure community details during run-time.

Remarks

None.

TCPIP_SNMP_MODULE_CONFIG Structure

SNMP module configuration.

File

snmpv3.h

C

```
typedef struct {
  bool trapEnable;
  bool snmp_trapv2_use;
  bool snmpv3_trapv1v2_use;
  char* snmp_bib_file;
  TCPIP_SNMP_COMMUNITY_CONFIG * read_community_config;
  TCPIP_SNMP_COMMUNITY_CONFIG * write_community_config;
  TCPIP_SNMPV3_USM_USER_CONFIG * usm_config;
  TCPIP_SNMPV3_TARGET_ENTRY_CONFIG * trap_target_config;
} TCPIP_SNMP_MODULE_CONFIG;
```

Members

Members	Description
bool trapEnable;	true = agent can send the trap, false = agent shouldn't send the trap
bool snmp_trapv2_use;	true = agent uses Trap version v2 and false = uses Tarp version 1
bool snmpv3_trapv1v2_use;	SNMPv3 trap should be true , only if SNMP version is 3
TCPIP_SNMP_COMMUNITY_CONFIG * read_community_config;	read-only Community configuration
TCPIP_SNMP_COMMUNITY_CONFIG * write_community_config;	write-only Community configuration
TCPIP_SNMPV3_USM_USER_CONFIG * usm_config;	SNMPv3 USM configuration
TCPIP_SNMPV3_TARGET_ENTRY_CONFIG * trap_target_config;	SNMPv3 trap configuration

Description

Structure: TCPIP_SNMP_MODULE_CONFIG

This structure is used to configure SNMP details for runtime configuration.

Remarks

None

TCPIP_SNMPV3_TARGET_ENTRY_CONFIG Structure

SNMP module trap target address configuration.

File

snmpv3.h

C

```
typedef struct {
  char * secname;
  STD_BASED_SNMP_SECURITY_MODEL mp_model;
```

```
  STD_BASED_SNMP_SECURITY_MODEL sec_model;
  STD_BASED_SNMPV3_SECURITY_LEVEL sec_level;
} TCPIP_SNMPV3_TARGET_ENTRY_CONFIG;
```

Description

Structure: SNMPV3_TARGET_ENTRY_CONFIG

This structure is used to configure SNMP target details during runtime .

Remarks

None.

TCPIP_SNMPV3_USM_USER_CONFIG Structure

SNMPv3 USM configuration.

File

snmpv3.h

C

```
typedef struct {
  char * username;
  STD_BASED_SNMPV3_SECURITY_LEVEL security_level;
  SNMPV3_HMAC_HASH_TYPE usm_auth_proto;
  char * usm_auth_password;
  SNMPV3_PRIV_PROT_TYPE usm_priv_proto;
  char * usm_priv_password;
} TCPIP_SNMPV3_USM_USER_CONFIG;
```

Members

Members	Description
char * username;	< user name string
STD_BASED_SNMPV3_SECURITY_LEVEL security_level;	< security level: auth, priv combination
SNMPV3_HMAC_HASH_TYPE usm_auth_proto;	< auth type: md5, sha1
char * usm_auth_password;	< passphrase string for authentication
SNMPV3_PRIV_PROT_TYPE usm_priv_proto;	< priv type: DES
char * usm_priv_password;	< passphrase string for privacy

Description

Structure: SNMPV3_USM_USER_CONFIG

This structure is used to configure predefined SNMPv3 USM details for run-time configuration.

Remarks

None.

Files

Files

Name	Description
snmp.h	Simple Network Management Protocol(SNMP) v1/v2c API header file.
snmpv3.h	Simple Network Management Protocol Version3(SNMPv3) API header file.
snmp_config.h	SNMP configuration file
snmpv3_config.h	SNMPv3 configuration file

Description

This section lists the source and header files used by the library.

snmp.h

Simple Network Management Protocol(SNMP) v1/v2c API header file.

Enumerations

	Name	Description
	SNMP_COMMUNITY_TYPE	Definition to represent different type of SNMP communities.
	SNMP_GENERIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP generic trap types.
	SNMP_TRAP_IP_ADDRESS_TYPE	Definition of the supported address types for SNMP trap.
	SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP vendor trap types.

Functions

	Name	Description
	TCPIP_SNMP_AuthTrapFlagGet	Gets the status of authentication trap flag.
	TCPIP_SNMP_AuthTrapFlagSet	Sets the status of authentication trap flag.
	TCPIP_SNMP_ClientGetNet	Get a network interface for SNMP TRAP.
	TCPIP_SNMP_ExactIndexGet	To search for exact index node in case of a Sequence variable.
	TCPIP_SNMP_IsTrapEnabled	Gets the SNMP Trap status.
	TCPIP_SNMP_IsValidCommunity	Validates community name for access control.
	TCPIP_SNMP_IsValidLength	Validates the set variable data length to data type.
	TCPIP_SNMP_MibIDSet	Sets the agent MIB ID for SNP notification.
	TCPIP_SNMP_NextIndexGet	To search for next index node in case of a Sequence variable.
	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_ReadCommunityGet	Gets the readCommunity String with SNMP index.
	TCPIP_SNMP_ReadCommunitySet	Sets the readCommunity String with SNMP index.
	TCPIP_SNMP_RecordIDValidation	Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.
	TCPIP_SNMP_SendFailureTrap	Prepares and validates the remote node that will receive a trap and send the trap PDU.
	TCPIP_SNMP_SocketIDGet	Gets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_SocketIDSet	Sets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_Task	Standard TCP/IP stack module task function.
	TCPIP_SNMP_TrapInterFaceSet	Sets the TRAP interface for SNMP notification.
	TCPIP_SNMP_TRAPMibIDGet	Gets the agent MIB ID for SNP notification.
	TCPIP_SNMP_TrapSendFlagGet	Gets the status of trap send flag.
	TCPIP_SNMP_TrapSendFlagSet	Sets the status of trap send flag.
	TCPIP_SNMP_TrapSpecificNotificationGet	Gets the specific trap.
	TCPIP_SNMP_TrapSpecificNotificationSet	Sets the specific trap, generic trap, and trap ID.
	TCPIP_SNMP_TrapTimeGet	Gets SNMP Trap UDP client open socket time-out.
	TCPIP_SNMP_TRAPTypeGet	Get SNMP Trap type for version v1 and v2c.
	TCPIP_SNMP_TRAPv1Notify	Creates and Sends TRAPv1 pdu.
	TCPIP_SNMP_TRAPv2Notify	Creates and sends TRAP PDU.
	TCPIP_SNMP_ValidateTrapIntf	Gets the status of trap interface.
	TCPIP_SNMP_VarbindGet	Used to get/collect OID variable information.
	TCPIP_SNMP_VarbindSet	Sets the MIB variable with the requested value.
	TCPIP_SNMP_WriteCommunityGet	Gets the writeCommunity String with SNMP index.
	TCPIP_SNMP_WriteCommunitySet	Sets the writeCommunity String with SNMP index.
	TCPIP_SNMPV3_TrapTypeGet	Gets SNMP Trap type for version v3.

Macros

	Name	Description
	SNMP_END_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.
	SNMP_INDEX_INVALID	This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.

	SNMP_V1	This macro is used for SNMP version 1
	SNMP_V2C	This macro is used for SNMP version 2 with community
	SNMP_V3	This macro is used for SNMP version 3 with authentication and privacy

Structures

	Name	Description
	SNMP_NON_MIB_RECD_INFO	Restrict access for specific OIDs.

Types

	Name	Description
	SNMP_ID	SNMP dynamic variable ID.
	SNMP_INDEX	SNMP sequence variable index.

Unions

	Name	Description
	SNMP_VAL	Definition to represent SNMP OID object values.

Description

SNMP Definitions for the Microchip TCP/IP Stack

Simple Network Management Protocol (SNMP) is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite. The SNMP Manager uses the UDP Port Number 161 to send requests to the agent. The agent uses the UDP Port Number 162 to send Trap(s) or notification events to the manager.

- Supports SNMP Version V1 and V2c over User Datagram Protocol (UDP)
- Supports Get, Get_Bulk, Get_Next, Set and Trap Protocol Data Units (PDUs)
- Supports up to 255 dynamic OIDs and unlimited constant OIDs
- mib2bib.jar utility is Microchip MIB compiler. It accepts Microchip MIB script

in ASCII format and generates two output files: The binary information file, snmp.bib and the C header file , mib.h.

File Name

snmp.h

Company

Microchip Technology Inc.

snmpv3.h

Simple Network Management Protocol Version3(SNMPv3) API header file.

Enumerations

	Name	Description
	SNMPV3_HMAC_HASH_TYPE	Different type of authentication for SNMPv3.
	SNMPV3_PRIV_PROT_TYPE	Different type of encryption and decryption for SNMPv3.
	STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL	Different SNMP Message processing model
	STD_BASED_SNMP_SECURITY_MODEL	Different Security services for SNMPv3 messages.
	STD_BASED_SNMPV3_SECURITY_LEVEL	Different Security Level for SNMPv3 messages.
	TCPIP_SNMPV3_USERDATABASECONFIG_TYPE	Different Configuration parameters of SNMPv3 operation

Functions

	Name	Description
	TCPIP_SNMPV3_EngineUserDataBaseGet	Get SNMPv3 engine data base details.
	TCPIP_SNMPV3_EngineUserDataBaseSet	Set SNMPv3 engine data base details.
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP PDU.
	TCPIP_SNMPv3_TrapConfigDataGet	Gets the SNMPv3 Trap configuration details using the user name index.

Structures

	Name	Description
	TCPIP_SNMP_COMMUNITY_CONFIG	SNMP community configuration.

	TCPIP_SNMP_MODULE_CONFIG	SNMP module configuration.
	TCPIP_SNMPV3_TARGET_ENTRY_CONFIG	SNMP module trap target address configuration.
	TCPIP_SNMPV3_USM_USER_CONFIG	SNMPv3 USM configuration.

Description

Simple Network Management Protocol (SNMP) v3 API Header File

SNMPv3 provides a secure access to the devices using a combination of authentication and encryption of packets over the network. Comparing with SNMPv1/v2, SNMPv3 ensures that packets can be collected securely from SNMP devices. Microchip SNMPv3 Agent authentication is implemented with MD5 and SHA1 protocols and encryption is implemented with AES protocol. These credentials and other user information are stored in the global array. The user of the SNMPv3 stack can decide on the number of user names in the Userâ€™s data base to be stored with the Server. According to the SNMPv3 recommendation, SNMPv3 server should not be configured with the authentication and privacy passwords. Instead could be configured with the respective localized keys of the password. Microchip SNMPv3 agent is provided with the password information in the database for the â€œGetting Startedâ€• and for understanding purpose only. It is recommended that the SNMPv3 stack should be modified to restrict access to the password OIDs declared in the user data base.

File Name

snmpv3.h

Company

Microchip Technology Inc.

snmp_config.h

SNMP configuration file

Macros

	Name	Description
	TCPIP_SNMP_BIB_FILE_NAME	The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.
	TCPIP_SNMP_COMMUNITY_MAX_LEN	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
	TCPIP_SNMP_MAX_COMMUNITY_SUPPORT	Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	TCPIP_SNMP_MAX_MSG_SIZE	The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16
	TCPIP_SNMP_MAX_NON_REC_ID_OID	Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it... more
	TCPIP_SNMP_NOTIFY_COMMUNITY_LEN	Maximum length for SNMP Trap community name
	TCPIP_SNMP_OID_MAX_LEN	Maximum length for the OID String. Change this to match your OID string length.
	TCPIP_SNMP_TASK_PROCESS_RATE	SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN	The maximum size of TRAP community string length

	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	Trap information. This macro will be used to avoid SNMP OID memory buffer corruption
	TCPIP_SNMP_TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

Description

Simple Network Management Protocol (SNMP) Configuration file

This file contains the SNMP module configuration options

File Name

snmp_config.h

Company

Microchip Technology Inc.

snmpv3_config.h

SNMPv3 configuration file

Macros

	Name	Description
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Authentication Localized password key length size
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Privacy Password key length size
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN	Maximum size for SNMPv3 User Security Name length.
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	TCPIP_SNMPV3_USM_MAX_USER	Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Description

Simple Network Management Protocol (SNMPv3) Configuration file

This file contains the SNMPv3 module configuration options

File Name

snmpv3_config.h

Company

Microchip Technology Inc.

SNTP Module

This section describes the TCP/IP Stack Library SNTP module.

Introduction

TCP/IP Stack Library Simple Network Time Protocol (SNTP) Module for Microchip Microcontrollers

This library provides the API of the SNTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to SNTPGetUTCSeconds using the internal Tick timer module.

The SNTP module is good for providing absolute time stamps. However, it should not be relied upon for measuring time differences (especially small differences). The pool of public time servers is implemented using round-robin DNS, so each update will come from a different server. Differing network delays and the fact that these servers are not verified implies that this time could be non-linear. While it is deemed reliable, it is not guaranteed to be accurate.

The Tick module provides much better accuracy (since it is driven by a hardware clock) and resolution, and should be used for measuring timeouts and other internal requirements.

Developers can change the value of NTP_SERVER if they want to always point to a preferred time server, or to specify a region when accessing time servers. The default is to use the global pool.

Using the Library

This topic describes the basic architecture of the SNTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `sntp.h`

The interface to the SNTP TCP/IP Stack library is defined in the `sntp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SNTP TCP/IP Stack library should include `tcpip.h`.

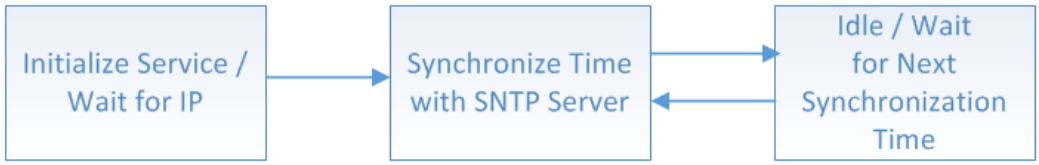
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SNTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNTP 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 SNTP module.

Library Interface Section	Description
Functions	Routines for configuring SNTP
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_NTP_DEFAULT_CONNECTION_TYPE	The default connection type to use: IPv4/IPv6
	TCPIP_NTP_DEFAULT_IF	for multi-homed hosts, the default SNTP interface
	TCPIP_NTP_EPOCH	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	TCPIP_NTP_FAST_QUERY_INTERVAL	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	TCPIP_NTP_MAX_STRATUM	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	TCPIP_NTP_QUERY_INTERVAL	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	TCPIP_NTP_REPLY_TIMEOUT	Defines how long to wait before assuming the query has failed, seconds
	TCPIP_NTP_RX_QUEUE_LIMIT	The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue
	TCPIP_NTP_SERVER	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too
	TCPIP_NTP_SERVER_MAX_LENGTH	maximum number of characters allowed for the NTP server
	TCPIP_NTP_TASK_TICK_RATE	THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_NTP_TIME_STAMP_TMO	elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL
	TCPIP_NTP_VERSION	The default NTP version to use (3 or 4)

Description

The configuration of the SNTP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SNTP TCP/IP Stack Library. Based on the selections made, the SNTP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SNTP TCP/IP Stack 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.

TCPIP_NTP_DEFAULT_CONNECTION_TYPE Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_DEFAULT_CONNECTION_TYPE (IP_ADDRESS_TYPE_IPV4)
```

Description

The default connection type to use: IPv4/IPv6

TCPIP_NTP_DEFAULT_IF Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_DEFAULT_IF "PIC32INT"
```

Description

for multi-homed hosts, the default SNTP interface

TCPIP_NTP_EPOCH Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_EPOCH (86400ul * (365ul * 70ul + 17ul))
```

Description

Reference Epoch to use. (default: 01-Jan-1970 00:00:00)

TCPIP_NTP_FAST_QUERY_INTERVAL Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_FAST_QUERY_INTERVAL (14ul)
```

Description

Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.

TCPIP_NTP_MAX_STRATUM Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_MAX_STRATUM (15)
```

Description

The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)

TCPIP_NTP_QUERY_INTERVAL Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_QUERY_INTERVAL (10ul*60ul)
```

Description

Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)

TCPIP_NTP_REPLY_TIMEOUT Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_REPLY_TIMEOUT (6ul)
```

Description

Defines how long to wait before assuming the query has failed, seconds

TCPIP_NTP_RX_QUEUE_LIMIT Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_RX_QUEUE_LIMIT (2)
```

Description

The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue

TCPIP_NTP_SERVER Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_SERVER "pool.ntp.org"
```

Description

These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too

TCPIP_NTP_SERVER_MAX_LENGTH Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_SERVER_MAX_LENGTH 30
```

Description

maximum number of characters allowed for the NTP server

TCPIP_NTP_TASK_TICK_RATE Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_TASK_TICK_RATE (1100)
```

Description

THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_NTP_TIME_STAMP_TMO Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_TIME_STAMP_TMO (11 * 60)
```

Description

elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL

TCPIP_NTP_VERSION Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_VERSION (4)
```

Description

The default NTP version to use (3 or 4)

Building the Library

This section lists the files that are available in the SNTP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/sntp.c`	SNTP 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 SNTP module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- UDP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server.
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server.
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters.

	TCPIP_SNTP_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_SNTP_MODULE_CONFIG	Placeholder for SNTP Module Configuration
	TCPIP_SNTP_RESULT	Provides a list of possible results for the SNTP module.

Description

This section describes the Application Programming Interface (API) functions of the SNTP module.
Refer to each section for a detailed description.

a) Functions

TCPIP_SNTP_ConnectionInitiate Function

Forces a connection to the NTP server

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionInitiate();
```

Returns

- SNTP_RES_OK - if the call succeeded
- SNTP_RES_PROGRESS - if a connection already in progress another one could not be started

Description

This function will start a connection to the NTP server

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionInitiate(void);

TCPIP_SNTP_LastErrorGet Function

Gets the last error code set in the NTP server.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_LastErrorGet();
```

Returns

The last error code encountered by the NTP module.

Description

This function returns the last NTP error code and clears the current error code.

Remarks

None/

Preconditions

The TCP/IP Stack should have been initialized.

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_LastErrorGet(void);

TCPIP_SNTP_UTCSecondsGet Function

Obtains the current time from the SNTP module.

File

sntp.h

C

```
uint32_t TCPIP_SNTP_UTCSecondsGet();
```

Returns

The number of seconds since the Epoch. (Default 01-Jan-1970 00:00:00)

Description

This function obtains the current time as reported by the SNTP module. Use this value for absolute time stamping. The value returned is (by default) the number of seconds since 01-Jan-1970 00:00:00.

Remarks

Do not use this function for time difference measurements. The Tick module is more appropriate for those requirements.

Preconditions

The TCP/IP Stack should have been initialized.

Function

uint32_t TCPIP_SNTP_UTCSecondsGet(void)

TCPIP_SNTP_TimeStampGet Function

Gets the last valid timestamp obtained from an NTP server.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_TimeStampGet(uint64_t* pTStamp, uint32_t* pLastUpdate);
```

Returns

- SNTP_RES_OK - if the call succeeded
- SNTP_RES_TSTAMP_STALE error code - if there is no recent timestamp

Description

This function gets the last valid timestamp obtained from an NTP server.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
pTStamp	pointer to a 64 bit buffer to store the last NTP timestamp could be NULL if the timestamp not needed
pLastUpdate	pointer to store the last time stamp update tick could be NULL if the update time not needed

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_TimeStampGet(uint64_t* pTStamp, uint32_t* pLastUpdate);

TCPIP_SNTP_ConnectionParamSet Function

Sets the current SNTP connection parameters.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionParamSet(TCPIP_NET_HANDLE netH, IP_ADDRESS_TYPE ntpConnType, const char* ntpServer);
```

Returns

- SNTP_RES_OK - if the call succeeded
- TCPIP_SNTP_RESULT error code - if the call did not succeed

Description

This function sets the parameters for the next SNTP connections.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
netH	new interface to use as default SNTP interface if 0, the current interface is not changed
ntpConnType	type of connection to make: IPv4 or IPv6 if IP_ADDRESS_TYPE_ANY, the current setting is not changed
ntpServer	the NTP server to be used; name or an IP address can be used if 0, the current NTP server is not changed

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionParamSet(TCPIP_NET_HANDLE netH, IP_ADDRESS_TYPE ntpConnType, const char* ntpServer);

TCPIP_SNTP_Task Function

Standard TCP/IP stack module task function.

File

sntp.h

C

```
void TCPIP_SNTP_Task();
```

Returns

None.

Description

This function performs SNTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The SNTP module should have been initialized.

Function

void TCPIP_SNTP_Task(void)

b) Data Types and Constants

TCPIP_SNTP_MODULE_CONFIG Structure

File

sntp.h

C

```
typedef struct {
  const char* ntp_server;
  const char* ntp_interface;
  IP_ADDRESS_TYPE ntp_connection_type;
  uint32_t ntp_reply_timeout;
  uint32_t ntp_stamp_timeout;
  uint32_t ntp_success_interval;
  uint32_t ntp_error_interval;
} TCPIP_SNTP_MODULE_CONFIG;
```

Members

Members	Description
const char* ntp_server;	the NTP server to contact; name or IP address
const char* ntp_interface;	the default NTP interface to use
IP_ADDRESS_TYPE ntp_connection_type;	the IPv4/IPv6 connection type
uint32_t ntp_reply_timeout;	timeout for the server reply in seconds
uint32_t ntp_stamp_timeout;	timeout for the time stamp in seconds
uint32_t ntp_success_interval;	server query interval after an successful update, in seconds
uint32_t ntp_error_interval;	server query interval after an error, in seconds

Description

Placeholder for SNTP Module Configuration

TCPIP_SNTP_RESULT Enumeration

Provides a list of possible results for the SNTP module.

File

sntp.h

C

```
typedef enum {
  SNTP_RES_OK,
  SNTP_RES_PROGRESS,
  SNTP_RES_BUSY = -1,
  SNTP_RES_TSTAMP_STALE = -2,
  SNTP_RES_SKT_ERR = -3,
  SNTP_RES_SKT_BIND_ERR = -4,
  SNTP_RES_NTP_SERVER_TMO = -5,
  SNTP_RES_NTP_VERSION_ERR = -6,
  SNTP_RES_NTP_TSTAMP_ERR = -7,
  SNTP_RES_NTP_SYNC_ERR = -8,
  SNTP_RES_NTP_KOD_ERR = -9,
  SNTP_RES_NTP_DNS_ERR = -10,
  SNTP_RES_NTP_IF_ERR = -11,
  SNTP_RES_NTP_CONN_ERR = -12
} TCPIP_SNTP_RESULT;
```

Members

Members	Description
SNTP_RES_OK	the operation was successful
SNTP_RES_PROGRESS	an NTP operation is in progress
SNTP_RES_BUSY = -1	module is busy

SNTP_RES_TSTAMP_STALE = -2	timestamp is stale, there's no recent timestamp
SNTP_RES_SKT_ERR = -3	NTP socket could not be opened
SNTP_RES_SKT_BIND_ERR = -4	NTP socket bind failed
SNTP_RES_NTP_SERVER_TMO = -5	NTP server could not be accessed
SNTP_RES_NTP_VERSION_ERR = -6	wrong NTP version received
SNTP_RES_NTP_TSTAMP_ERR = -7	wrong NTP time stamp received
SNTP_RES_NTP_SYNC_ERR = -8	NTP time synchronization error
SNTP_RES_NTP_KOD_ERR = -9	an NTP KissOfDeath code has been received
SNTP_RES_NTP_DNS_ERR = -10	an NTP DNS error
SNTP_RES_NTP_IF_ERR = -11	an NTP interface error
SNTP_RES_NTP_CONN_ERR = -12	an NTP connection type error

Description

TCPIP_SNTP_RESULT Enumeration

Provides a list of possible SNTP results.

Remarks

None

Files

Files

Name	Description
sntp.h	The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers.
sntp_config.h	SNTP configuration file

Description

This section lists the source and header files used by the library.

sntp.h

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers.

Enumerations

	Name	Description
	TCPIP_SNTP_RESULT	Provides a list of possible results for the SNTP module.

Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters.
	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server.
	TCPIP_SNTP_Task	Standard TCP/IP stack module task function.
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server.
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.

Structures

	Name	Description
	TCPIP_SNTP_MODULE_CONFIG	Placeholder for SNTP Module Configuration

Description

SNTP Client Module API Header File

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to SNTPGetUTCSeconds using the internal Tick timer module.

File Name

sntp.h

Company

Microchip Technology Inc.

sntp_config.h

SNTP configuration file

Macros

	Name	Description
	TCPIP_NTP_DEFAULT_CONNECTION_TYPE	The default connection type to use: IPv4/IPv6
	TCPIP_NTP_DEFAULT_IF	for multi-homed hosts, the default SNTP interface
	TCPIP_NTP_EPOCH	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	TCPIP_NTP_FAST_QUERY_INTERVAL	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	TCPIP_NTP_MAX_STRATUM	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	TCPIP_NTP_QUERY_INTERVAL	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	TCPIP_NTP_REPLY_TIMEOUT	Defines how long to wait before assuming the query has failed, seconds
	TCPIP_NTP_RX_QUEUE_LIMIT	The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue
	TCPIP_NTP_SERVER	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too
	TCPIP_NTP_SERVER_MAX_LENGTH	maximum number of characters allowed for the NTP server
	TCPIP_NTP_TASK_TICK_RATE	THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_NTP_TIME_STAMP_TMO	elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL
	TCPIP_NTP_VERSION	The default NTP version to use (3 or 4)

Description

Network Time Protocol (SNTP) Configuration file

This file contains the SNTP module configuration options

File Name

sntp_config.h

Company

Microchip Technology Inc.

TCP Module

This section describes the TCP/IP Stack Library TCP module.

Introduction

TCP/IP Stack Library Transmission Control Protocol (TCP) Module for Microchip Microcontrollers

This library provides the API of the TCP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

TCP is a standard transport layer protocol described in RFC 793. It provides reliable stream-based connections over unreliable networks, and forms the foundation for HTTP, SMTP, and many other protocol standards.

Using the Library

This topic describes the basic architecture of the TCP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcp.h`

The interface to the TCP TCP/IP Stack library is defined in the `tcp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the TCP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

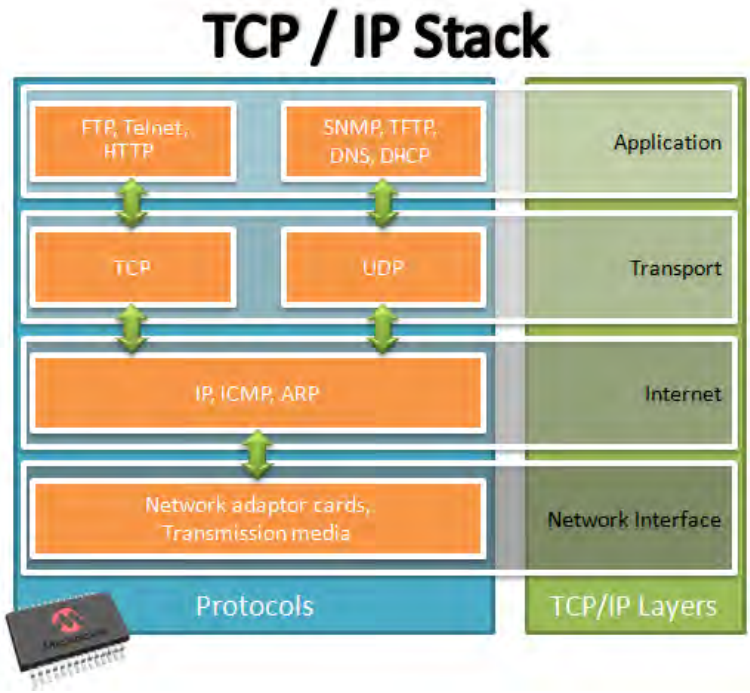
This library provides the API of the TCP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

TCP Software Abstraction Block Diagram

This module provides software abstraction of the TCP module existent in any TCP/IP Stack implementation.

Typical TCP Implementation



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the TCP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing TCP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions
Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

How the Library Works

This topic describes how the TCP/IP Stack Library TCP module works.

Description

Connections made over TCP guarantee data transfer at the expense of throughput. Connections are made through a three-way handshake process, ensuring a one-to-one connection. Remote nodes advertise how much data they are ready to receive, and all data transmitted must be acknowledged. If a remote node fails to acknowledge the receipt of data, it is automatically retransmitted. This ensures that network errors such as lost, corrupted, or out-of-order packets are automatically corrected.

To accomplish this, TCP must use buffers for both the transmit and receive operations. Once the transmit buffer is full, no more data can be sent until the remote node has acknowledged receipt. For the Microchip TCP/IP Stack, the application must return to the main system loop to allow the other services in the system, including the TCP/IP stack, to advance.. Likewise, the remote node cannot transmit more data until the local device has acknowledged receipt and that space is available in the buffer. When a local application needs to read more data, it must return to the main system/application loop and wait for a new packet to arrive.

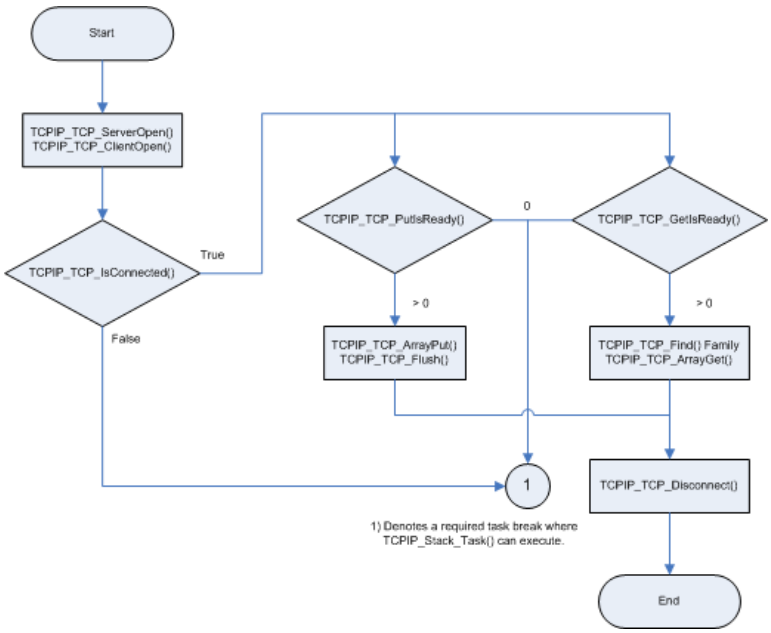
Core Functionality

This topic describes the core functionality of the TCP/IP Stack Library TCP module.

Description

The following diagram provides an overview for the use of the TCP module.

TCP Connection Flow



Server sockets are opened using TCPIP_TCP_ServerOpen. This function will open a listening socket waiting for client connections on the specified TCP port number. A client socket is opened using TCPIP_TCP_ClientOpen. This function will open a socket that connects to a remote host. The remote host is specified using an IP address and a TCP port number.

Once connected, applications can read and write data. On each entry, the application must verify that the socket is still connected. For most applications a call to TCPIP_TCP_IsConnected will be sufficient, but TCPIP_TCP_WasReset may also be used for listening sockets that may turn over quickly.

To write data, call TCPIP_TCP_PutIsReady to check how much space is available. Then, call the TCPIP_TCP_ArrayPut function to write data as space is available. Once complete, call TCPIP_TCP_Flush to transmit data immediately. Alternately, return to the main system/stack loop. Data will be transmitted by the internal TCP state machine as the buffer becomes full or a programmable delay time has passed.

To read data, call TCPIP_TCP_GetIsReady to determine how many bytes are ready to be retrieved. Then use the TCPIP_TCP_ArrayGet function to read data from the socket, and/or the TCPIP_TCP_Find family of functions to locate data in the buffer. When no more data remains, return to the main system/stack loop to wait for more data to arrive.

If the application needs to close the connection, call TCPIP_TCP_Disconnect, and then return to the main system/stack loop and wait for the remote node to acknowledge the disconnection. Client sockets will be closed and associated resources freed, while listening sockets will wait for a new connection. For more information, refer to the examples provided with the MPLAB Harmony distribution examples, or read the associated RFC.

Configuring the Library

Macros

	Name	Description
	TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL	Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.
	TCPIP_TCP_CLOSE_WAIT_TIMEOUT	Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).
	TCPIP_TCP_DELAYED_ACK_TIMEOUT	Timeout for delayed-acknowledgment algorithm, ms
	TCPIP_TCP_FIN_WAIT_2_TIMEOUT	Timeout for FIN WAIT 2 state, ms
	TCPIP_TCP_KEEP_ALIVE_TIMEOUT	Timeout for keep-alive messages when no traffic is sent, ms
	TCPIP_TCP_MAX_RETRIES	Maximum number of retransmission attempts
	TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. Maximum Segment Size for RX (MSS)... more
	TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL	Maximum Segment Size for RX (MSS) for non local destination networks.
	TCPIP_TCP_MAX_SEG_SIZE_TX	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	TCPIP_TCP_MAX_SOCKETS	The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.
	TCPIP_TCP_MAX_SYN_RETRIES	Smaller than all other retries to reduce SYN flood DoS duration
	TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE	Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_START_TIMEOUT_VAL	Timeout to retransmit unacked data, ms

	TCPIP_TCP_TASK_TICK_RATE	The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL	Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.
	TCPIP_TCP_DYNAMIC_OPTIONS	Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets
	TCPIP_TCP_MSL_TIMEOUT	Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.
	TCPIP_TCP_QUIET_TIME	This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.

Description

The configuration of the TCP/IP Stack Library TCP module is based on the file `system_config.h` (which may include `tcp_config.h`).

This header file contains the configuration selection for the TCP/IP Stack Library TCP module. Based on the selections made, the TCP/IP Stack Library TCP module may support the selected features. These configuration settings will apply to all TCP sockets of the TCP/IP Stack 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.

TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL (40ul)
```

Description

Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.

TCPIP_TCP_CLOSE_WAIT_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_CLOSE_WAIT_TIMEOUT (200ul)
```

Description

Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).

TCPIP_TCP_DELAYED_ACK_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_DELAYED_ACK_TIMEOUT (100ul)
```

Description

Timeout for delayed-acknowledgment algorithm, ms

TCPIP_TCP_FIN_WAIT_2_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_FIN_WAIT_2_TIMEOUT (5000ul)
```

Description

Timeout for FIN WAIT 2 state, ms

TCPIP_TCP_KEEP_ALIVE_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_KEEP_ALIVE_TIMEOUT (10000ul)
```

Description

Timeout for keep-alive messages when no traffic is sent, ms

TCPIP_TCP_MAX_RETRIES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_RETRIES (5u)
```

Description

Maximum number of retransmission attempts

TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL (1460)
```

Description

TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network.

Maximum Segment Size for RX (MSS) for local destination networks.

TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL (536)
```

Description

Maximum Segment Size for RX (MSS) for non local destination networks.

TCPIP_TCP_MAX_SEG_SIZE_TX Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_TX (1460u)
```

Description

TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.

TCPIP_TCP_MAX_SOCKETS Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SOCKETS (10)
```

Description

The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.

TCPIP_TCP_MAX_SYN_RETRIES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SYN_RETRIES (2u)
```

Description

Smaller than all other retries to reduce SYN flood DoS duration

TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES (6u)
```

Description

Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection

TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE 512
```

Description

Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to specify the buffer size for their TCP sockets.

TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to specify the buffer size for their TCP sockets.

TCPIP_TCP_START_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_START_TIMEOUT_VAL (1000ul)
```

Description

Timeout to retransmit unacked data, ms

TCPIP_TCP_TASK_TICK_RATE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_TASK_TICK_RATE (5)
```

Description

The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL (200ul)
```

Description

Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.

TCPIP_TCP_DYNAMIC_OPTIONS Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_DYNAMIC_OPTIONS 1
```

Description

Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets

TCPIP_TCP_MSL_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MSL_TIMEOUT (30)
```

Description

Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.

TCPIP_TCP_QUIET_TIME Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_QUIET_TIME (30)
```

Description

This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.

Building the Library

This section lists the files that are available in the TCP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/tcp.c`	TCP 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 TCP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Socket Management Functions

	Name	Description
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.
	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket.
	TCPIP_TCP_Bind	Binds a socket to a local address.
	TCPIP_TCP_RemoteBind	Binds a socket to a remote address.
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.
	TCPIP_TCP_Disconnect	Disconnects an open socket.
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current RX/TX buffer size, etc.
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket.
	TCPIP_TCP_SocketNetGet	Gets the current network interface of an TCP socket.
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket
	TCPIP_TCP_SignalHandlerDeregister	Deregisters a previously registered TCP socket signal handler.
	TCPIP_TCP_SignalHandlerRegister	Registers a TCP socket signal handler.
	TCPIP_TCP_Task	Standard TCP/IP stack module task function.
	TCPIP_TCP_IsReady	Returns the current status of the TCP module
	TCPIP_TCP_SocketsNumberGet	Returns the number of existent TCP sockets.
	TCPIP_TCP_SocketTraceSet	Sets the current socket trace status.

b) Transmit Data Transfer Functions

	Name	Description
	TCPIP_TCP_Put	Writes a single byte to a TCP socket.
	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
	TCPIP_TCP_StringPut	Writes a null-terminated string to a TCP socket.
	TCPIP_TCP_ArrayPut	Writes an array from a buffer to a TCP socket.
	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
	TCPIP_TCP_FifoTxFreeGet	Determines how many bytes are free and could be written in the TCP TX FIFO.

c) Receive Data Transfer Functions

	Name	Description
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket's RX buffer/FIFO.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket.
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.
	TCPIP_TCP_Discard	Discards any pending data in the RCP RX FIFO.
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX buffer/FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_FifoRxFullGet	Determines how many bytes are pending in the RX buffer/FIFO.

d) Data Types and Constants

	Name	Description
	TCP_ADJUST_FLAGS	TCP adjust RX and TX buffers flags.
	TCP_OPTION_LINGER_DATA	Socket linger options.
	TCP_PORT	Defines a TCP Port number.
	TCP_SOCKET	Defines a TCP Socket.
	TCP_SOCKET_INFO	TCP socket information.
	TCP_SOCKET_OPTION	TCP Socket run-time options.
	INVALID_SOCKET	Invalid socket indicator macro.
	TCP_OPTION_THRES_FLUSH_TYPE	List of the socket half threshold TX flush types.
	TCPIP_TCP_MODULE_CONFIG	TCP module run-time configuration/initialization data.
	TCP_OPTION_KEEP_ALIVE_DATA	Socket keep alive options
	TCPIP_TCP_SIGNAL_FUNCTION	TCP signal handler.
	TCPIP_TCP_SIGNAL_HANDLE	TCP socket handle.
	TCPIP_TCP_SIGNAL_TYPE	TCP run time signal/event types.
	TCPIP_TCP_STATE	TCP socket state.

Description

This section describes the Application Programming Interface (API) functions of the TCP module.

Refer to each section for a detailed description.

a) Socket Management Functions

TCPIP_TCP_ServerOpen Function

Opens a TCP socket as a server.

File

tcp.h

C

```
TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened
- TCP_SOCKET handle - Save this handle and use it when calling all other TCP APIs

Description

Provides a unified method for opening TCP server sockets.

Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP_TCP_Close.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT localPort	TCP port to listen on for connections.
IP_MULTI_ADDRESS* localAddress	Local address to use. Use 0 if listening on any network interface.

Function

TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_TCP_ClientOpen Function

Opens a TCP socket as a client.

File

tcp.h

C

```
TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened
- TCP_SOCKET handle - Save this handle and use it when calling all other TCP APIs

Description

Provides a unified method for opening TCP client sockets.

Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP_TCP_Abort or TCPIP_TCP_Close.

Remarks

IP_ADDRESS_TYPE_ANY is not supported (not a valid type for client open)!

If the remoteAddress != 0 (and the address pointed by remoteAddress != 0) then the socket will immediately initiate a connection to the remote host

If the remoteAddress is unspecified, no connection is initiated. Client socket parameters can be set using TCPIP_TCP_Bind, TCPIP_TCP_RemoteBind, etc. calls and then connection initiated by calling TCPIP_TCP_Connect.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT remotePort	TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to be used. If 0 then the address is unspecified

Function

TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_TCP_Close Function

Disconnects an open socket and destroys the socket handle, releasing the associated resources.

File

tcp.h

C

```
bool TCPIP_TCP_Close(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - If the call failed (no such socket)

Description

Graceful Option Set: If the graceful option is set for the socket (default), a TCPIP_TCP_Disconnect will be tried. If the linger option is set (default) the TCPIP_TCP_Disconnect will try to send any queued TX data before issuing FIN. If the FIN send operation fails or the socket is not connected the abort is generated.

Graceful Option Not Set: If the graceful option is not set, or the previous step could not send the FIN,a TCPIP_TCP_Abort is called, sending a RST to the remote node. Communication is closed, the socket is no longer valid and the associated resources are freed.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen/TCPIP_TCP_ClientOpen. hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect and close.

Function

bool TCPIP_TCP_Close(TCP_SOCKET hTCP)

TCPIP_TCP_Connect Function

Connects a client socket.

File

tcp.h

C

```
bool TCPIP_TCP_Connect(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - If the call failed

Description

This function will try to initiate a connection on a client socket that is not connected yet. The client socket should have been created with a call to TCPIP_TCP_ClientOpen having the remoteAddress set to 0.

Remarks

The call will fail if the client socket has no remote host specified. Use TCPIP_TCP_RemoteBind() to specify a remote host address for the client socket.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen with an unspecified address. hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the client socket to connect.

Function

bool TCPIP_TCP_Connect(TCP_SOCKET hTCP)

TCPIP_TCP_Bind Function

Binds a socket to a local address.

File

tcp.h

C

```
bool TCPIP_TCP_Bind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function is meant for unconnected server and client sockets. It is similar to TCPIP_TCP_SocketNetSet that assigns a specific source interface for a socket. If localPort is 0 the stack will assign a unique local port. Sockets don't need specific binding, it is done automatically by the stack. However, specific binding can be requested using these functions. Works for both client and server sockets.

Remarks

The call will fail if the socket is already connected (both server and client sockets).

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
localPort	Local port to use If 0, the stack will assign a unique local port
localAddress	The local address to bind to. Could be NULL if the local address does not need to be changed

Function

bool TCPIP_TCP_Bind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_TCP_RemoteBind Function

Binds a socket to a remote address.

File

tcp.h

C

```
bool TCPIP_TCP_RemoteBind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function is meant for unconnected server and client sockets. Sockets don't need specific remote binding, they should accept connections on any incoming interface. Thus the binding is done automatically by the stack. However, specific remote binding can be requested using these functions. For a server socket it can be used to restrict accepting connections from a specific remote host. For a client socket it will just change the default binding done when the socket was opened.

Remarks

The socket remote port is changed only if a non-zero remotePort value is passed.

The socket remote host address is changed only if a non-zero remoteAddress value is passed.

The call will fail if the socket is already connected (both server and client sockets).

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
remotePort	remote port to use Could be 0 if the remote port does not need to be changed
remoteAddress	The remote address to bind to. Could be NULL if the remote address does not need to be changed

Function

bool TCPIP_TCP_RemoteBind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_TCP_IsConnected Function

Determines if a socket has an established connection.

File

tcp.h

C

```
bool TCPIP_TCP_IsConnected(TCP_SOCKET hTCP);
```

Description

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen() to determine when the connection is set up and ready for use.

Remarks

A socket is said to be connected only if it is in the TCPIP_TCP_STATE_ESTABLISHED state. Sockets in the process of opening or closing will return false.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_IsConnected(TCP_SOCKET hTCP)

TCPIP_TCP_WasReset Function

Self-clearing semaphore indicating socket reset.

File

tcp.h

C

```
bool TCPIP_TCP_WasReset(TCP_SOCKET hTCP);
```

Description

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function works for all possible disconnections: a call to TCPIP_TCP_Disconnect, a FIN from the remote node, or an acknowledgment timeout caused by the loss of a network link. It also returns true after the first call to TCPIP_TCP_Initialize. Applications should use this function to reset their state machines.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_WasReset(TCP_SOCKET hTCP)

TCPIP_TCP_Disconnect Function

Disconnects an open socket.

File

tcp.h

C

```
bool TCPIP_TCP_Disconnect(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - Indicates that the notification could not be sent to the remote host. The call can be reissued at a later time if desired.

Description

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection.

If the socket has the linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the linger option is off, the queued TX data will be discarded.

Please note that this call may fail in which case it can be re-issued.

Remarks

None.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle of the socket to disconnect.

Function

bool TCPIP_TCP_Disconnect(TCP_SOCKET hTCP)

TCPIP_TCP_Abort Function

Aborts a connection.

File

tcp.h

C

```
void TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket);
```

Returns

None.

Description

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client socket will always be closed and the associated resources released. The socket cannot be used again after this call.

A server socket will abort the current connection:

- if killSocket == false the socket will remain listening
- if killSocket == true the socket will be closed and all associated resources released. The socket cannot be used again after this call.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect.
killSocket	if true, it kills a server socket

Function

void TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket)

TCPIP_TCP_OptionsGet Function

Allows getting the options for a socket like: current RX/TX buffer size, etc.

File

tcp.h

C

```
bool TCPIP_TCP_OptionsGet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be get at the socket level. This function provides compatibility with BSD implementations.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent the size of the area has to be large enough to accommodate the specific option • TCP_OPTION_LINGER - pointer to a TCP_OPTION_LINGER_DATA structure • TCP_OPTION_KEEP_ALIVE - pointer to a TCP_OPTION_KEEP_ALIVE_DATA structure • TCP_OPTION_RX_BUFF - size of the new RX buffer • TCP_OPTION_TX_BUFF - size of the new TX buffer • TCP_OPTION_RX_TMO - not supported yet • TCP_OPTION_TX_TMO - not supported yet • TCP_OPTION_NODELAY - pointer to boolean to return current NO DELAY status • TCP_OPTION_EXCLUSIVE_ADDRESS - not supported yet • TCP_OPTION_THRES_FLUSH - a TCP_OPTION_THRES_FLUSH_TYPE • TCP_OPTION_DELAY_SEND_ALL_ACK - pointer to boolean to return current DELAY Send All ACK status

Function

bool TCPIP_TCP_OptionsGet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam)

TCPIP_TCP_OptionsSet Function

Allows setting options to a socket like adjust RX/TX buffer size, etc.

File

tcp.h

C

```
bool TCPIP_TCP_OptionsSet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	socket to set options for
option	specific option to be set
optParam	the option value; this is option dependent • TCP_OPTION_LINGER - pointer to a TCP_OPTION_LINGER_DATA structure • TCP_OPTION_KEEP_ALIVE - pointer to a TCP_OPTION_KEEP_ALIVE_DATA structure • TCP_OPTION_RX_BUFF - size of the new RX buffer • TCP_OPTION_TX_BUFF - size of the new TX buffer • TCP_OPTION_RX_TMO - not supported yet • TCP_OPTION_TX_TMO - not supported yet • TCP_OPTION_NODELAY - boolean to enable/disable the NO DELAY functionality • TCP_OPTION_EXCLUSIVE_ADDRESS - not supported yet • TCP_OPTION_THRES_FLUSH - a TCP_OPTION_THRES_FLUSH_TYPE • TCP_OPTION_DELAY_SEND_ALL_ACK - boolean to enable/disable the DELAY Send All ACK data functionality

Function

bool TCPIP_TCP_OptionsSet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam)

TCPIP_TCP_SocketInfoGet Function

Obtains information about a currently open socket.

File

tcp.h

C

```
bool TCPIP_TCP_SocketInfoGet(TCP_SOCKET hTCP, TCP_SOCKET_INFO* pInfo);
```

Returns

- true - if the call succeeded
- false - if no such socket exists or the socket is not open

Description

Fills the provided TCP_SOCKET_INFO structure associated with this socket. This contains the IP addresses and port numbers for both the local and remote endpoints.

Preconditions

TCP is initialized and the socket is connected.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_SocketInfoGet(TCP_SOCKET hTCP, TCP_SOCKET_INFO* remoteInfo)

TCPIP_TCP_SocketNetGet Function

Gets the current network interface of an TCP socket.

File

tcp.h

C

```
TCPIP_NET_HANDLE TCPIP_TCP_SocketNetGet(TCP_SOCKET hTCP);
```

Returns

The handle of the local interface this socket is bound to.

Description

This function returns the interface handle associated to a TCP socket.

Remarks

The returned handle could be NULL if the socket is invalid or the socket is not currently connected.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen()/TCPIP_TCP_ServerOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The TCP socket

Function

TCPIP_NET_HANDLE TCPIP_TCP_SocketNetGet(TCP_SOCKET hTCP);

TCPIP_TCP_SocketNetSet Function

Sets the interface for an TCP socket

File

tcp.h

C

```
bool TCPIP_TCP_SocketNetSet(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function sets the network interface for an TCP socket

Remarks

An invalid hNet can be passed (0) so that the current network interface selection will be cleared.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen()/TCPIP_TCP_ServerOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The TCP socket
hNet	interface handle.

Function

bool TCPIP_TCP_SocketNetSet(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet)

TCPIP_TCP_SignalHandlerDeregister Function

Deregisters a previously registered TCP socket signal handler.

File

tcp.h

C

```
bool TCPIP_TCP_SignalHandlerDeregister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_HANDLE hSig);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the TCP socket signal handler.

Preconditions

hSig valid TCP handle.

Parameters

Parameters	Description
s	The TCP socket
hSig	A handle returned by a previous call to TCPIP_TCP_SignalHandlerRegister.

Function

TCPIP_TCP_SignalHandlerDeregister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_HANDLE hSig)

TCPIP_TCP_SignalHandlerRegister Function

Registers a TCP socket signal handler.

File

tcp.h

C

```
TCPIP_TCP_SIGNAL_HANDLE TCPIP_TCP_SignalHandlerRegister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_TYPE sigMask,
TCPIP_TCP_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (null handler, no such socket, existent handler).

Description

This function registers a TCP socket signal handler. The TCP module will call the registered handler when a TCP signal (TCPIP_TCP_SIGNAL_TYPE) occurs.

Remarks

Only one signal handler per socket is supported. A new handler does not override the existent one. Instead TCPIP_TCP_SignalHandlerDeregister has to be explicitly called.

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client but is currently not used and should be 0.

For multi-threaded systems the TCP/IP packet dispatch does not occur on the user thread. The signal handler will be called on a different thread context. It is essential that this handler is non blocking and really fast.

For multi-threaded systems, once set, it is not recommended to change the signal handler at run time. Synchronization between user threads and packet dispatch threads could be difficult. If really need to be changed, make sure that the old handler could still be called and it should be valid until the new one is taken into account. TCPIP_TCP_SignalHandlerDeregister needs to be called before registering another handler.

Preconditions

TCP valid socket.

Parameters

Parameters	Description
s	The TCP socket
sigMask	mask of signals to be reported
handler	signal handler to be called when a TCP event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the TCP module. Currently not used and it should be null.

Function

TCPIP_TCP_SignalHandlerRegister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_TYPE sigMask,
TCPIP_TCP_SIGNAL_FUNCTION handler, const void* hParam)

TCPIP_TCP_Task Function

Standard TCP/IP stack module task function.

File

tcp.h

C

```
void TCPIP_TCP_Task();
```

Returns

None.

Description

This function performs TCP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP module should have been initialized.

Function

void TCPIP_TCP_Task(void)

TCPIP_TCP_IsReady Function

Returns the current status of the TCP module

File

tcp.h

C

```
bool TCPIP_TCP_IsReady();
```

Returns

true - the quiet time has elapsed and the TCP module is ready to transmit data false - the quiet time is still in effect and TCP module is silent

Description

This function returns the current status of the TCP module. If the TCP quiet time has elapsed (or it was not enabled when TCP was built) then the TCP module is ready for operation and TCP sockets can send and receive data. If the TCP quiet time is still in effect, then the TCP module is quiet and it won't transmit any kind of data.

Remarks

The TCP quiet time occurs at the stack start up and can be enabled/disabled using the TCP module configuration parameters.

Preconditions

TCP module properly initialized

Function

bool TCPIP_TCP_IsReady(void);

TCPIP_TCP_SocketsNumberGet Function

Returns the number of existent TCP sockets.

File

tcp.h

C

```
int TCPIP_TCP_SocketsNumberGet();
```

Returns

The number of TCP sockets

Description

This function returns the number of created TCP sockets. This is the maximum number of sockets that can be opened at any moment as it's been passed as parameter when TCP module was created.

Preconditions

TCP module properly initialized

Function

int TCPIP_TCP_SocketsNumberGet(void)

TCPIP_TCP_SocketTraceSet Function

Sets the current socket trace status.

File

tcp.h

C

```
bool TCPIP_TCP_SocketTraceSet(TCP_SOCKET sktNo, bool enable);
```

Returns

- true if the call succeeded
- false if there was an error (no such socket, tracing not enabled, etc.)

Description

This function enables or disables the trace status of the specified socket. The trace functionality needs to be enabled in the TCP module for this function to succeed. Currently when socket trace is enabled, the current socket state transitions are displayed at the system console.

Preconditions

TCP module properly initialized

Parameters

Parameters	Description
sktNo	socket to enable/disable trace
enable	boolean to enable/disable socket trace

Function

bool TCPIP_TCP_SocketTraceSet(TCP_SOCKET sktNo, bool enable)

b) Transmit Data Transfer Functions

TCPIP_TCP_Put Function

File

tcp.h

C

```
uint16_t TCPIP_TCP_Put(TCP_SOCKET hTCP, uint8_t byte);
```

Description

Writes a single byte to a TCP socket.

Remarks

Note that this function is inefficient and its use is discouraged. A buffered approach (TCPIP_TCP_ArrayPut) is preferred.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
byte	The byte to write.

Function

uint16_t TCPIP_TCP_Put(TCP_SOCKET hTCP, uint8_t byte)

TCPIP_TCP_PutIsReady Function

Determines how much free space is available in the TCP TX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_PutIsReady(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be written in the TCP TX buffer.

Description

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_PutIsReady(TCP_SOCKET hTCP)

TCPIP_TCP_StringPut Function

Writes a null-terminated string to a TCP socket.

File

tcp.h

C

```
const uint8_t* TCPIP_TCP_StringPut(TCP_SOCKET hTCP, const uint8_t* Data);
```

Returns

Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NULL byte, the buffer became full or the socket is not connected.

Description

This function writes a null-terminated string to a TCP socket. The null-terminator is not copied to the socket.

Remarks

The return value of this function differs from that of TCPIP_TCP_ArrayPut. To write long strings in a single state, initialize the *data pointer to the first byte, then call this function repeatedly (breaking to the main stack loop after each call) until the return value dereferences to a NULL byte. Save the return value as the new starting *data pointer otherwise.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the string to be written.

Function

const uint8_t* TCPIP_TCP_StringPut(TCP_SOCKET hTCP, const uint8_t* data)

TCPIP_TCP_ArrayPut Function

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayPut(TCP_SOCKET hTCP, const uint8_t* Data, uint16_t Len);
```

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

Description

Writes an array from a buffer to a TCP socket.

Remarks

This operation can cause a TCP packet to be transmitted over the network (i.e., a TCPIP_TCP_Flush operation to be performed) when there is data in the TCP TX buffer that can be sent and any of the following occurs:

- There is no more space available in the TCP buffer.
- The TCP_OPTION_THRES_FLUSH_ON is set and the TX buffer is at least half full
- The amount of data that can be sent is bigger than the remote host MSS or than half of the maximum advertised window size.
- The Nagle algorithm is disabled and there is no unacknowledged data.

If none of these occur and the socket user does not add data to the TX socket buffer, the TCP state machine will automatically flush the buffer when the TCP_AUTO_TRANSMIT_TIMEOUT_VAL time-out elapsed.

The default TCP_AUTO_TRANSMIT_TIMEOUT_VAL is 40 ms.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the array to be written.
len	Number of bytes to be written.

Function

uint16_t TCPIP_TCP_ArrayPut(TCP_SOCKET hTCP, const uint8_t* data, uint16_t len)

TCPIP_TCP_Flush Function

Immediately transmits all pending TX data.

File

tcp.h

C

```
bool TCPIP_TCP_Flush(TCP_SOCKET hTCP);
```

Returns

None,

Description

This function immediately transmits all pending TX data with a PSH flag. If this function is not called, data will automatically be sent by the TCP state machine (see the TCPIP_TCP_ArrayPut() description/notes).

Remarks

The application should not call this function explicitly because doing this will interfere with the TCP algorithm and degrade performance of the socket data transfer. One exception is when the application knows that it put all the data it needed into the TCP buffer and it makes sense to flush the socket instead of waiting TCP_AUTO_TRANSMIT_TIMEOUT_VAL timeout to elapse.

Preconditions

TCP is initialized and the socket is connected.

Parameters

Parameters	Description
hTCP	The socket whose data is to be transmitted.

Function

bool TCPIP_TCP_Flush(TCP_SOCKET hTCP)

TCPIP_TCP_FifoTxFullGet Function

Determines how many bytes are pending in the TCP TX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoTxFullGet(TCP_SOCKET hTCP);
```

Returns

Number of bytes pending to be flushed in the TCP TX FIFO.

Description

This function determines how many bytes are pending in the TCP TX FIFO.

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_FifoTxFullGet(TCP_SOCKET hTCP)

TCPIP_TCP_FifoTxFreeGet Function

Determines how many bytes are free and could be written in the TCP TX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoTxFreeGet(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be written in the TCP TX buffer.

Description

Macro: TCPIP_TCP_FifoTxFreeGet(TCP_SOCKET hTCP)

This macro returns the number of bytes that are free in the socket TX buffer.

Remarks

None.

Parameters

Parameters	Description
hTCP	The socket to check.

c) Receive Data Transfer Functions

TCPIP_TCP_GetIsReady Function

Determines how many bytes can be read from the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_GetIsReady(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be read from the TCP RX buffer.

Description

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_GetIsReady(TCP_SOCKET hTCP)

TCPIP_TCP_ArrayGet Function

Reads an array of data bytes from a TCP socket's RX buffer/FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayGet(TCP_SOCKET hTCP, uint8_t* buffer, uint16_t count);
```

Returns

The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected.

Description

This function reads an array of data bytes from a TCP socket's RX buffer/FIFO. The data is removed from the FIFO in the process.

Remarks

If the supplied buffer is null, the data is simply discarded.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket from which data is to be read.
buffer	Pointer to the array to store data that was read.
len	Number of bytes to be read.

Function

uint16_t TCPIP_TCP_ArrayGet(TCP_SOCKET hTCP, uint8_t* buffer, uint16_t len)

TCPIP_TCP_ArrayPeek Function

Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayPeek(TCP_SOCKET hTCP, uint8_t * vBuffer, uint16_t wLen, uint16_t wStart);
```

Description

This function reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer. No TCP control actions are taken as a result of this function (ex: no window update is sent to the remote node).

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peek from (read without removing from stream).
vBuffer	Destination to write the peeked data bytes.
wLen	Length of bytes to peek from the RX FIFO and copy to vBuffer.
wStart	Zero-indexed starting position within the FIFO from which to start peeking.

Function

uint16_t TCPIP_TCP_ArrayPeek(TCP_SOCKET hTCP, uint8_t *vBuffer, uint16_t wLen, uint16_t wStart)

TCPIP_TCP_ArrayFind Function

Searches for a string in the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayFind(TCP_SOCKET hTCP, const uint8_t* cFindArray, uint16_t wLen, uint16_t wStart,
uint16_t wSearchLen, bool bTextCompare);
```

Description

This function finds the first occurrence of an array of bytes in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the search array is "love" with a length of 4, a value of 2 will be returned.

Remarks

For better performance of this function, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to an internal buffer for comparison. This has proven to be significantly faster than searching for full header name strings outright.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFindArray	The array of bytes to find in the buffer.
wLen	Length of cFindArray.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Function

uint16_t TCPIP_TCP_ArrayFind(TCP_SOCKET hTCP, uint8_t* cFindArray, uint16_t wLen,

uint16_t wStart, uint16_t wSearchLen, bool bTextCompare)

TCPIP_TCP_Find Function

Searches for a byte in the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Find(TCP_SOCKET hTCP, uint8_t cFind, uint16_t wStart, uint16_t wSearchLen, bool
bTextCompare);
```

Description

This function finds the first occurrence of a byte in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the cFind byte is ' ', a value of 1 will be returned.

Remarks

For better performance of this function, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to an internal buffer for comparison. This has proven to be significantly faster than searching for full header name strings outright.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFind	The byte to find in the buffer.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Function

uint16_t TCPIP_TCP_Find(TCP_SOCKET hTCP, uint8_t cFind, uint16_t wStart,
uint16_t wSearchLen, bool bTextCompare)

TCPIP_TCP_Get Function

Retrieves a single byte to a TCP socket.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Get(TCP_SOCKET hTCP, uint8_t* byte);
```

Description

This function retrieves a single byte to a TCP socket.

Remarks

Note that this function is inefficient. A buffered approach (TCPIP_TCP_ArrayGet) is preferred.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket from which to read.
byte	Pointer to location in which the read byte should be stored.

Function

uint16_t TCPIP_TCP_Get(TCP_SOCKET hTCP, uint8_t* byte)

TCPIP_TCP_Peek Function

Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.

File

tcp.h

C

```
uint8_t TCPIP_TCP_Peek(TCP_SOCKET hTCP, uint16_t wStart);
```

Description

This function peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.

Remarks

Note that this function is inefficient and its use is discouraged. A buffered approach (TCPIP_TCP_ArrayPeek) is preferred.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peek from (read without removing from stream).
wStart	Zero-indexed starting position within the FIFO to peek from.

Function

uint8_t TCPIP_TCP_Peek(TCP_SOCKET hTCP, uint16_t wStart)

TCPIP_TCP_Discard Function

Discards any pending data in the RCP RX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Discard(TCP_SOCKET hTCP);
```

Returns

The number of bytes that have been discarded from the RX buffer.

Description

This function discards any pending data in the TCP RX FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket whose RX buffer/FIFO is to be cleared.

Function

uint16_t TCPIP_TCP_Discard(TCP_SOCKET hTCP)

TCPIP_TCP_FifoRxFreeGet Function

Determines how many bytes are free in the RX buffer/FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoRxFreeGet(TCP_SOCKET hTCP);
```

Returns

The number of bytes free in the TCP RX FIFO. If zero, no additional data can be received until the application removes some data using one of the TCPIP_TCP_Get family functions.

Description

This function determines how many bytes are free in the RX buffer/FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_FifoRxFreeGet(TCP_SOCKET hTCP)

TCPIP_TCP_FifoSizeAdjust Function

Adjusts the relative sizes of the RX and TX buffers.

File

tcp.h

C

```
bool TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize, TCP_ADJUST_FLAGS vFlags);
```

Description

This function can be used to simultaneously adjust the sizes of the RX and TX FIFOs.

Adjusting the size of the TX/RX FIFO on the fly can allow for optimal transmission speed for one-sided application protocols. For example, HTTP typically begins by receiving large amounts of data from the client, then switches to serving large amounts of data back. Adjusting the FIFO at these points can increase performance in systems that have limited resources. Once the FIFOs are adjusted, a window update is sent.

Although the TX and RX socket buffers are completely independent, for the purpose of this function they can be considered together if neither TCP_ADJUST_TX_ONLY nor TCP_ADJUST_RX_ONLY flags are set.

In these conditions the TCP_ADJUST_FLAGS control the distribution of the available space between the TX and RX FIFOs. If neither or both of TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are set, the function distributes the remaining space (if any) equally. If the new requested FIFOs space is greater that the old existing FIFOs space the TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are ignored.

TCP_ADJUST_PRESERVE_RX and TCP_ADJUST_PRESERVE_TX request the preserving of the existing data. Existing data can be preserved as long as the old data in the buffer does not exceed the capacity of the new buffer.

Remarks

The function will automatically maintain minimal values for both TX and RX buffers.

To avoid having a socket with no associated buffers, the function first allocates the new buffers and, if succeeded, it frees the old ones.

The TX and RX FIFOs (buffers) associated with a socket are completely separate and independent. However, when TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY are not set, for the purpose of this function, the TX and RX FIFOs are considered to be contiguous so that the total FIFO space is divided between the TX and RX FIFOs. This provides backward compatibility with previous versions of this function.

The new flags TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY allow changing the size of TX and RX buffers independently. This is the preferred option.

TX or RX associated buffer sizes can be changed too using the socket options. See TCPIP_TCP_OptionsSet.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to be adjusted
wMinRXSize	Minimum number of bytes for the RX FIFO
wMinTXSize	Minimum number of bytes for the TX FIFO
vFlags	If TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY are not set, the TX and RX buffers are evaluated together and any
combination of the following flags is valid	TCP_ADJUST_GIVE_REST_TO_RX, TCP_ADJUST_GIVE_REST_TO_TX TCP_ADJUST_PRESERVE_RX, TCP_ADJUST_PRESERVE_TX If TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY is set TX and RX buffers are treated individually and TCP_ADJUST_GIVE_REST_TO_TX, TCP_ADJUST_GIVE_REST_TO_RX values are irrelevant. TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY both set is invalid.

Function

bool TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize,
TCP_ADJUST_FLAGS vFlags)

TCPIP_TCP_FifoRxFullGet Macro

Determines how many bytes are pending in the RX buffer/FIFO.

File

tcp.h

C

```
#define TCPIP_TCP_FifoRxFullGet(a) TCPIP_TCP_GetIsReady(a)
```

Description

Macro: TCPIP_TCP_FifoRxFullGet

Alias to TCPIP_TCP_GetIsReady provided for API completeness

Remarks

None.

d) Data Types and Constants

TCP_ADJUST_FLAGS Enumeration

TCP adjust RX and TX buffers flags.

File

tcp.h

C

```
typedef enum {
  TCP_ADJUST_GIVE_REST_TO_RX = 0x01,
  TCP_ADJUST_GIVE_REST_TO_TX = 0x02,
  TCP_ADJUST_PRESERVE_RX = 0x04,
  TCP_ADJUST_PRESERVE_TX = 0x08,
  TCP_ADJUST_TX_ONLY = 0x10,
  TCP_ADJUST_RX_ONLY = 0x20
} TCP_ADJUST_FLAGS;
```

Members

Members	Description
TCP_ADJUST_GIVE_REST_TO_RX = 0x01	Resize flag: extra bytes go to RX
TCP_ADJUST_GIVE_REST_TO_TX = 0x02	Resize flag: extra bytes go to TX
TCP_ADJUST_PRESERVE_RX = 0x04	Resize flag: attempt to preserve RX buffer
TCP_ADJUST_PRESERVE_TX = 0x08	Resize flag: attempt to preserve TX buffer
TCP_ADJUST_TX_ONLY = 0x10	Resize flag: adjust the TX buffer only
TCP_ADJUST_RX_ONLY = 0x20	Resize flag: adjust the RX buffer only

Description

Enumeration: TCP_ADJUST_FLAGS

Adjusts socket RX and TX buffer sizes.

TCP_OPTION_LINGER_DATA Structure

Socket linger options.

File

tcp.h

C

```
typedef struct {
  bool lingerEnable;
  bool gracefulEnable;
  uint16_t lingerTmo;
} TCP_OPTION_LINGER_DATA;
```

Members

Members	Description
bool lingerEnable;	Enable/disable linger; enabled by default for any socket.
bool gracefulEnable;	Enable/disable graceful close; enabled by default for any socket.
uint16_t lingerTmo;	Linger timeout in seconds (when enabled). This option is not supported yet.

Description

Structure: TCP_OPTION_LINGER_DATA

This structure defines socket linger options.

TCP_PORT Type

Defines a TCP Port number.

File

tcp.h

C

```
typedef uint16_t TCP_PORT;
```

Description

Type: TCP_PORT

TCP Port Number identifier.

TCP_SOCKET Type

Defines a TCP Socket.

File

tcp.h

C

```
typedef int16_t TCP_SOCKET;
```

Description

Type: TCP_SOCKET

A TCP_SOCKET is stored as a single int16_t number.

TCP_SOCKET_INFO Structure

TCP socket information.

File

tcp.h

C

```
typedef struct {
  IP_ADDRESS_TYPE addressType;
  IP_MULTI_ADDRESS remoteIPaddress;
  IP_MULTI_ADDRESS localIPaddress;
  TCP_PORT remotePort;
  TCP_PORT localPort;
  TCPIP_NET_HANDLE hNet;
  TCPIP_TCP_STATE state;
  uint16_t rxSize;
  uint16_t txSize;
  uint16_t rxPending;
  uint16_t txPending;
} TCP_SOCKET_INFO;
```

Members

Members	Description
IP_ADDRESS_TYPE addressType;	Address type of the socket IPv4 or IPv6
IP_MULTI_ADDRESS remoteIPaddress;	Remote address to which the socket is connected
IP_MULTI_ADDRESS localIPaddress;	Local address socket is bound to
TCP_PORT remotePort;	Port number associated with remote node
TCP_PORT localPort;	Local port number
TCPIP_NET_HANDLE hNet;	Associated interface
TCPIP_TCP_STATE state;	Current socket state
uint16_t rxSize;	size of the RX buffer
uint16_t txSize;	size of the TX buffer
uint16_t rxPending;	bytes pending in RX buffer
uint16_t txPending;	bytes pending in TX buffer

Description

Structure: TCP_SOCKET_INFO

Gets information about a socket.

TCP_SOCKET_OPTION Enumeration

TCP Socket run-time options.

File

tcp.h

C

```
typedef enum {
  TCP_OPTION_LINGER,
  TCP_OPTION_KEEP_ALIVE,
  TCP_OPTION_RX_BUFF,
  TCP_OPTION_TX_BUFF,
  TCP_OPTION_RX_TMO,
  TCP_OPTION_TX_TMO,
  TCP_OPTION_NODELAY,
  TCP_OPTION_EXCLUSIVE_ADDRESS,
  TCP_OPTION_THRES_FLUSH,
  TCP_OPTION_DELAY_SEND_ALL_ACK
} TCP_SOCKET_OPTION;
```

Members

Members	Description
TCP_OPTION_LINGER	The LINGER option controls the action taken when unsent data is queued on a socket and the socket is closed. The linger option can be turned on/off and the timeout can be specified.
TCP_OPTION_KEEP_ALIVE	Enable the use of keep-alive packets on TCP connections. The option can be turned on/off and the timeout can be specified.
TCP_OPTION_RX_BUFF	Request different RX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated.
TCP_OPTION_TX_BUFF	Request different TX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated.
TCP_OPTION_RX_TMO	Specifies the RX timeout. If no data arrives in the specified timeout the socket is closed.
TCP_OPTION_TX_TMO	Specifies the TX timeout. If no data can be sent in the specified timeout the socket is closed.
TCP_OPTION_NODELAY	Enables the NO DELAY/Nagle algorithm functionality. The default setting is disabled.
TCP_OPTION_EXCLUSIVE_ADDRESS	Enables a socket to be bound for exclusive access. The default setting is disabled - option not supported yet.
TCP_OPTION_THRES_FLUSH	Sets the type of half buffer TX flush for the socket. The default setting is TCP_OPTION_THRES_FLUSH_AUTO

TCP_OPTION_DELAY_SEND_ALL_ACK	Enables/disables the delay of sending the pending data when no unacknowledged data. When this option is disabled, the socket will immediately send any pending data chunk whenever all the previous data is acknowledged by the remote party (there is no unacknowledged data). This holds true for the 1st transmission too, or after a pause of some length. If the option is enabled, the socket will delay sending the data, waiting for more data to accumulate. This could be useful when the application makes multiple calls with small data chunks. The default setting is disabled.

Description

Enumeration: TCP_SOCKET_OPTION

This enumeration defines TCP socket run-time options.

INVALID_SOCKET Macro

Invalid socket indicator macro.

File

tcp.h

C

```
#define INVALID_SOCKET (-1)
```

Description

Macro: INVALID_SOCKET

Indicates that the socket is invalid or could not be opened.

TCP_OPTION_THRES_FLUSH_TYPE Enumeration

List of the socket half threshold TX flush types.

File

tcp.h

C

```
typedef enum {
  TCP_OPTION_THRES_FLUSH_AUTO,
  TCP_OPTION_THRES_FLUSH_OFF,
  TCP_OPTION_THRES_FLUSH_ON
} TCP_OPTION_THRES_FLUSH_TYPE;
```

Members

Members	Description
TCP_OPTION_THRES_FLUSH_AUTO	The socket will set the half buffer flush based on the TX buffer size. This is the default setting. For TX buffers >= 1.5 MSS the half buffer flush will be disabled. This results in higher performance/throughput for the socket.
TCP_OPTION_THRES_FLUSH_OFF	Always disable the socket flush at half buffer threshold. No socket flush is performed when the half TX buffer threshold is passed.
TCP_OPTION_THRES_FLUSH_ON	Always enable the socket flush at half buffer threshold. Socket flush is always performed when the half TX buffer threshold is passed. This is useful for small TX buffers when the remote party implements the delayed ACK algorithm.

Description

Enumeration: TCP_OPTION_THRES_FLUSH_TYPE

Describes the possible types of the socket half threshold TX flush.

TCPIP_TCP_MODULE_CONFIG Structure

TCP module run-time configuration/initialization data.

File

tcp.h

C

```
typedef struct {
  int nSockets;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
} TCPIP_TCP_MODULE_CONFIG;
```

Members

Members	Description
int nSockets;	Number of sockets to be created
uint16_t sktTxBuffSize;	Size of the socket TX buffer
uint16_t sktRxBuffSize;	Size of the socket RX buffer

Description

Structure: TCPIP_TCP_MODULE_CONFIG

This structure defines TCP module run-time configuration/initialization data.

TCP_OPTION_KEEP_ALIVE_DATA Structure

Socket keep alive options

File

tcp.h

C

```
typedef struct {
  bool keepAliveEnable;
  uint16_t keepAliveTmo;
  uint8_t keepAliveUnackLim;
} TCP_OPTION_KEEP_ALIVE_DATA;
```

Members

Members	Description
bool keepAliveEnable;	Enable/disable keep alive option; disabled by default for any socket.
uint16_t keepAliveTmo;	keep alive timeout in milliseconds ignored when keep alive is disabled if 0, the default build time value is used
uint8_t keepAliveUnackLim;	limit of keep alives to be sent the socket will reset the communication channel if no reply received after so many retries ignored when keep alive is disabled if 0, the default build time value is used

Description

Structure: TCP_OPTION_KEEP_ALIVE_DATA

This structure defines socket keep alive options.

TCPIP_TCP_SIGNAL_FUNCTION Type

TCP signal handler.

File

tcp.h

C

```
typedef void (* TCPIP_TCP_SIGNAL_FUNCTION)(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet, TCPIP_TCP_SIGNAL_TYPE
sigType, const void* param);
```

Description

Type: TCPIP_TCP_SIGNAL_FUNCTION

Prototype of a TCP signal handler. Socket user can register a handler for the TCP socket. Once an TCP event occurs the registered handler will be called.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

Parameters

Parameters	Description
hTCP	TCP socket to be used
hNet	the network interface on which the event has occurred
sigType	type of TCP signal that has occurred
param	additional parameter that can has been specified at the handler registration call is currently not used and will be null

TCPIP_TCP_SIGNAL_HANDLE Type

TCP socket handle.

File

tcp.h

C

```
typedef const void* TCPIP_TCP_SIGNAL_HANDLE;
```

Description

Type: TCPIP_TCP_SIGNAL_HANDLE

A handle that a socket client can use after the signal handler has been registered.

TCPIP_TCP_SIGNAL_TYPE Enumeration

TCP run time signal/event types.

File

tcp.h

C

```
typedef enum {
  TCPIP_TCP_SIGNAL_TX_DONE = 0x0001,
  TCPIP_TCP_SIGNAL_TX_DATA_DONE = 0x0002,
  TCPIP_TCP_SIGNAL_TX_SPACE = 0x0004,
  TCPIP_TCP_SIGNAL_ESTABLISHED = 0x0100,
  TCPIP_TCP_SIGNAL_RX_DATA = 0x0200,
  TCPIP_TCP_SIGNAL_RX_FIN = 0x0400,
  TCPIP_TCP_SIGNAL_RX_RST = 0x0800,
  TCPIP_TCP_SIGNAL_KEEP_ALIVE_TMO = 0x1000
} TCPIP_TCP_SIGNAL_TYPE;
```

Members

Members	Description
TCPIP_TCP_SIGNAL_TX_DONE = 0x0001	A TCP packet was successfully transmitted on the interface. This may indicate or not that new data can be sent with this socket. Note 1: The TCP buffer space is controlled by the TCP state machine; The fact that a packet was transmitted, doesn't necessarily mean that new data can be written into the socket buffer: 2: The packet may be an ACK only packet, retry packet, etc., and does not free any writing space in the socket buffer 3: A new socket writing space may not be available because the other party has not acknowledged the data it was sent so far. Therefore, the information carried by this signal has limited usage. 4: This notification is the result of an user action (explicit call to TCPIP_TCP_Flush) or as a result of the internal TCP state machine
TCPIP_TCP_SIGNAL_TX_DATA_DONE = 0x0002	A TCP data packet carrying valid payload was successfully transmitted on the interface. This may indicate or not that new data can be sent with this socket.
TCPIP_TCP_SIGNAL_TX_SPACE = 0x0004	New TX space is available in the socket TX buffer. This event signals that the remote party has acknowledged some data and, as a result, TX buffer space is available.
TCPIP_TCP_SIGNAL_ESTABLISHED = 0x0100	Socket established a connection (client or server socket)
TCPIP_TCP_SIGNAL_RX_DATA = 0x0200	A data packet was successfully received and there is data available for this socket
TCPIP_TCP_SIGNAL_RX_FIN = 0x0400	Remote host finished it's data and sent a FIN;
TCPIP_TCP_SIGNAL_RX_RST = 0x0800	Remote host reset the connection;
TCPIP_TCP_SIGNAL_KEEP_ALIVE_TMO = 0x1000	Keep alive has timed out; Connection to the remote host has been aborted;

Description

Enumeration: TCPIP_TCP_SIGNAL_TYPE

Description of the signals/events that a TCP socket can generate.

Remarks

These signals are used in the socket event handling notification functions. It is possible that multiple flags are set as part of the same notification.

The signals are 16 bits wide.

TCPIP_TCP_STATE Enumeration

TCP socket state.

File

tcp.h

C

```
typedef enum {
  TCPIP_TCP_STATE_LISTEN,
  TCPIP_TCP_STATE_SYN_SENT,
  TCPIP_TCP_STATE_SYN_RECEIVED,
  TCPIP_TCP_STATE_ESTABLISHED,
  TCPIP_TCP_STATE_FIN_WAIT_1,
  TCPIP_TCP_STATE_FIN_WAIT_2,
  TCPIP_TCP_STATE_CLOSING,
  TCPIP_TCP_STATE_TIME_WAIT,
  TCPIP_TCP_STATE_CLOSE_WAIT,
  TCPIP_TCP_STATE_LAST_ACK,
  TCPIP_TCP_STATE_CLIENT_WAIT_DISCONNECT,
  TCPIP_TCP_STATE_CLIENT_WAIT_CONNECT,
  TCPIP_TCP_STATE_KILLED
} TCPIP_TCP_STATE;
```

Members

Members	Description
TCPIP_TCP_STATE_LISTEN	Socket is listening for connections
TCPIP_TCP_STATE_SYN_SENT	A SYN has been sent, awaiting an SYN+ACK
TCPIP_TCP_STATE_SYN_RECEIVED	A SYN has been received, awaiting an ACK
TCPIP_TCP_STATE_ESTABLISHED	Socket is connected and connection is established
TCPIP_TCP_STATE_FIN_WAIT_1	FIN WAIT state 1
TCPIP_TCP_STATE_FIN_WAIT_2	FIN WAIT state 2
TCPIP_TCP_STATE_CLOSING	Socket is closing
TCPIP_TCP_STATE_TIME_WAIT	2MSL state
TCPIP_TCP_STATE_CLOSE_WAIT	Waiting to close the socket
TCPIP_TCP_STATE_LAST_ACK	The final ACK has been sent
TCPIP_TCP_STATE_CLIENT_WAIT_DISCONNECT	client socket lost connection, waiting for app close/disconnect
TCPIP_TCP_STATE_CLIENT_WAIT_CONNECT	client socket waiting for connection
TCPIP_TCP_STATE_KILLED	socket is killed; debug reporting state

Description

Enumeration: TCPIP_TCP_STATE

Enumeration describing the possible states of a TCP socket.

Files

Files

Name	Description
tcp.h	Transmission Control Protocol (TCP) Communications Layer API
tcp_config.h	TCP configuration file

Description

This section lists the source and header files used by the library.

tcp.h

Transmission Control Protocol (TCP) Communications Layer API

Enumerations

	Name	Description
	TCP_ADJUST_FLAGS	TCP adjust RX and TX buffers flags.
	TCP_OPTION_THRES_FLUSH_TYPE	List of the socket half threshold TX flush types.
	TCP_SOCKET_OPTION	TCP Socket run-time options.
	TCPIP_TCP_SIGNAL_TYPE	TCP run time signal/event types.
	TCPIP_TCP_STATE	TCP socket state.

Functions

	Name	Description
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket's RX buffer/FIFO.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.
	TCPIP_TCP_ArrayPut	Writes an array from a buffer to a TCP socket.
	TCPIP_TCP_Bind	Binds a socket to a local address.
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.
	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket.
	TCPIP_TCP_Discard	Discards any pending data in the RCP RX FIFO.
	TCPIP_TCP_Disconnect	Disconnects an open socket.
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX buffer/FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_FifoTxFreeGet	Determines how many bytes are free and could be written in the TCP TX FIFO.
	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket.
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_IsReady	Returns the current status of the TCP module
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current RX/TX buffer size, etc.
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.
	TCPIP_TCP_Put	Writes a single byte to a TCP socket.
	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
	TCPIP_TCP_RemoteBind	Binds a socket to a remote address.
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_SignalHandlerDeregister	Deregisters a previously registered TCP socket signal handler.
	TCPIP_TCP_SignalHandlerRegister	Registers a TCP socket signal handler.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket.
	TCPIP_TCP_SocketNetGet	Gets the current network interface of an TCP socket.
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket
	TCPIP_TCP_SocketsNumberGet	Returns the number of existent TCP sockets.
	TCPIP_TCP_SocketTraceSet	Sets the current socket trace status.
	TCPIP_TCP_StringPut	Writes a null-terminated string to a TCP socket.

	TCPIP_TCP_Task	Standard TCP/IP stack module task function.
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.

Macros

	Name	Description
	INVALID_SOCKET	Invalid socket indicator macro.
	TCPIP_TCP_FifoRxFullGet	Determines how many bytes are pending in the RX buffer/FIFO.

Structures

	Name	Description
	TCP_OPTION_KEEP_ALIVE_DATA	Socket keep alive options
	TCP_OPTION_LINGER_DATA	Socket linger options.
	TCP_SOCKET_INFO	TCP socket information.
	TCPIP_TCP_MODULE_CONFIG	TCP module run-time configuration/initialization data.

Types

	Name	Description
	TCP_PORT	Defines a TCP Port number.
	TCP_SOCKET	Defines a TCP Socket.
	TCPIP_TCP_SIGNAL_FUNCTION	TCP signal handler.
	TCPIP_TCP_SIGNAL_HANDLE	TCP socket handle.

Description

Transmission Control Protocol (TCP) Communications Layer API Header File

- Provides reliable, handshaked transport of application stream oriented data with flow control
- Reference: RFC 793

File Name

tcp.h

Company

Microchip Technology Inc.

tcp_config.h

TCP configuration file

Macros

	Name	Description
	TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL	Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.
	TCPIP_TCP_CLOSE_WAIT_TIMEOUT	Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).
	TCPIP_TCP_DELAYED_ACK_TIMEOUT	Timeout for delayed-acknowledgment algorithm, ms
	TCPIP_TCP_DYNAMIC_OPTIONS	Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets
	TCPIP_TCP_FIN_WAIT_2_TIMEOUT	Timeout for FIN WAIT 2 state, ms
	TCPIP_TCP_KEEP_ALIVE_TIMEOUT	Timeout for keep-alive messages when no traffic is sent, ms
	TCPIP_TCP_MAX_RETRIES	Maximum number of retransmission attempts

	TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. Maximum Segment Size for RX (MSS)... more
	TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL	Maximum Segment Size for RX (MSS) for non local destination networks.
	TCPIP_TCP_MAX_SEG_SIZE_TX	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	TCPIP_TCP_MAX_SOCKETS	The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.
	TCPIP_TCP_MAX_SYN_RETRIES	Smaller than all other retries to reduce SYN flood DoS duration
	TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	TCPIP_TCP_MSL_TIMEOUT	Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.
	TCPIP_TCP_QUIET_TIME	This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.
	TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE	Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_START_TIMEOUT_VAL	Timeout to retransmit unacked data, ms
	TCPIP_TCP_TASK_TICK_RATE	The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL	Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.

Description

Transmission Control Protocol (TCP) Configuration file

This file contains the TCP module configuration options

File Name

tcp_config.h

Company

Microchip Technology Inc.

TFTP Module

This section describes the TCP/IP Stack Library TFTP module.

Introduction

This library provides the API of the TFTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

TCP/IP Stack Library Trivial File Transfer Protocol (TFTP) module is simple protocol used to transfer files.

The TFTP module only supports client transfers, through the process of reading and writing files (or mail) from/to a remote server. The module cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8-bit bytes of data.

Using the Library

This topic describes the basic architecture of the TFTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tftpc.h`

The interface to the TFTP TCP/IP Stack library is defined in the `tftpc.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the `tftpc.h` TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the TFTP module.

Library Interface Section	Description
General Functions	This section provides general interface routines to TFTP
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic provides information on how the TFTP module works including write and read requests.

Description

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes, signals termination of a transfer. If a packet gets lost in the network, the intended recipient will time out and may retransmit the last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet.

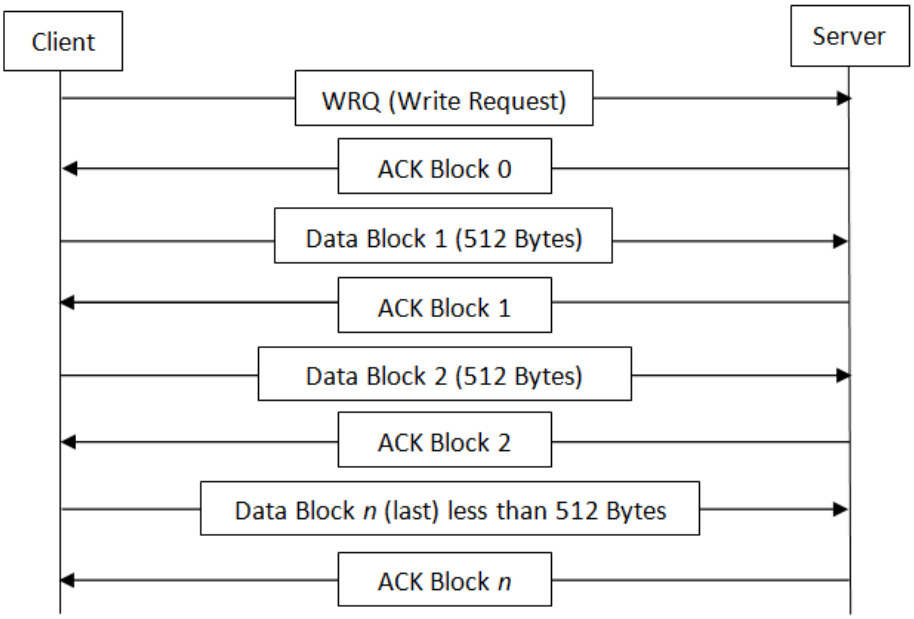
TFTP Client Server Communication

This section provides information on TFTP Write and Read requests.

Description

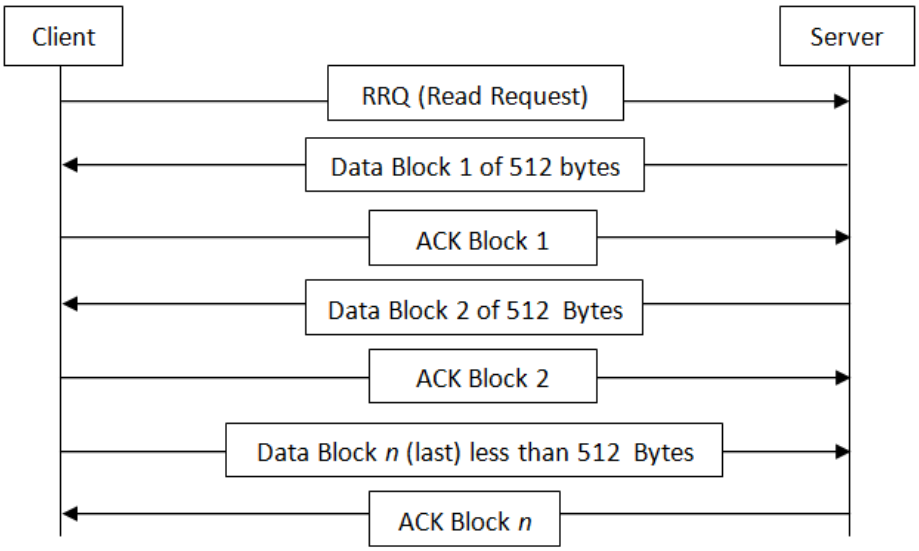
TFTP Write Request (WRQ)

The following diagram illustrates a write request.



TFTP Read Request (RRQ)

The following diagram illustrates a read request.



Configuring the Library

Macros

	Name	Description
	TCPIP_TFTPC_ARP_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on PUT.
	TCPIP_TFTPC_CMD_PROCESS_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on GET.
	TCPIP_TFTPC_DEFAULT_IF	The default TFTP interface for multi-homed hosts.
	TCPIP_TFTPC_FILENAME_LEN	The maximum value for the file name size.
	TCPIP_TFTPC_HOSTNAME_LEN	The maximum TFTP host name length size.
	TCPIP_TFTPC_MAX_RETRIES	The number of attempts before declaring a TIMEOUT error.
	TCPIP_TFTPC_TASK_TICK_RATE	The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

	TCPIP_TFTPC_USER_NOTIFICATION	allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Description

The configuration of the TFTP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the TFTP TCP/IP Stack Library. Based on the selections made, the TFTP TCP/IP Stack Library will or will not support selected features. These configuration settings will apply to all instances of the TFTP TCP/IP Stack 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.

TCPIP_TFTPC_ARP_TIMEOUT Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_ARP_TIMEOUT (3ul)
```

Description

The number of seconds to wait before declaring a TIMEOUT error on PUT.

TCPIP_TFTPC_CMD_PROCESS_TIMEOUT Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_CMD_PROCESS_TIMEOUT (3ul)
```

Description

The number of seconds to wait before declaring a TIMEOUT error on GET.

TCPIP_TFTPC_DEFAULT_IF Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_DEFAULT_IF "PIC32INT"
```

Description

The default TFTP interface for multi-homed hosts.

TCPIP_TFTPC_FILENAME_LEN Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_FILENAME_LEN (32)
```

Description

The maximum value for the file name size.

TCPIP_TFTPC_HOSTNAME_LEN Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_HOSTNAME_LEN (32)
```

Description

The maximum TFTP host name length size.

TCPIP_TFTPC_MAX_RETRIES Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_MAX_RETRIES (3u)
```

Description

The number of attempts before declaring a TIMEOUT error.

TCPIP_TFTPC_TASK_TICK_RATE Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_TASK_TICK_RATE (100)
```

Description

The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TFTPC_USER_NOTIFICATION Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_USER_NOTIFICATION false
```

Description

allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Building the Library

This section lists the files that are available in the TFTP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/tftpc.c`	TFTP 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 TFTP module depends on the following module:

- TCP/IP Stack Library

Library Interface

a) Functions

	Name	Description
	TCPIP_TFTPC_Task	Standard TCP/IP stack module task function.
	TCPIP_TFTPC_GetEventNotification	Get the details of the TFTP client file mode communication event details.
	TCPIP_TFTPC_HandlerDeRegister	Deregisters a previously registered TFTP Client handler.
	TCPIP_TFTPC_HandlerRegister	Registers a TFTPC Handler.
	TCPIP_TFTPC_SetServerAddress	Set the Server IP address for TFTP Client .
	TCPIP_TFTPC_SetCommand	TFTP client command operation configuration.

b) Data Types and Constants

	Name	Description
	_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTPC_MODULE_CONFIG	Placeholder for TFTP Client module configuration.
	TCPIP_TFTPC_EVENT_HANDLER	TFTPC event handler prototype.
	TCPIP_TFTPC_EVENT_TYPE	TFTP client Event Type
	TCPIP_TFTPC_HANDLE	TFTPC handle.
	TCPIP_TFTPC_OPERATION_RESULT	Standard error codes for TFTP PUT and GET command operation.

Description

This section describes the Application Programming Interface (API) functions of the TFTP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_TFTPC_Task Function

Standard TCP/IP stack module task function.

File

tftpc.h

C

```
void TCPIP_TFTPC_Task();
```

Returns

None.

Description

This function performs TFTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The TFTP module should have been initialized.

Function

void TCPIP_TFTPC_Task(void)

TCPIP_TFTPC_GetEventNotification Function

Get the details of the TFTP client file mode communication event details.

File

tftpc.h

C

```
TCPIP_TFTPC_EVENT_TYPE TCPIP_TFTPC_GetEventNotification();
```

Returns

TCPIP_TFTPC_EVENT_TYPE : It will be OR of different events.

Description

This function returns the event type TCPIP_TFTPC_EVENT_TYPE for different modes of TFTP file communication.

Preconditions

The TFTP Client module must be initialized.

Function

TCPIP_TFTPC_EVENT_TYPE TCPIP_TFTPC_GetEventNotification(void)

TCPIP_TFTPC_HandlerDeRegister Function

Deregisters a previously registered TFTP Client handler.

File

tftpc.h

C

```
bool TCPIP_TFTPC_HandlerDeRegister(TCPIP_TFTPC_HANDLE hDhcp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the TFTP Client event handler.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
htftpc	A handle returned by a previous call to TCPIP_TFTPC_HandlerRegister.

Function

bool TCPIP_TFTPC_HandlerDeRegister(TCPIP_TFTPC_HANDLE htftpc)

TCPIP_TFTPC_HandlerRegister Function

Registers a TFTPC Handler.

File

tftpc.h

C

```
TCPIP_TFTPC_HANDLE TCPIP_TFTPC_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_HANDLER handler,
const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

This function registers a TFTPC event handler. The TFTP Client module will call the registered handler when a TFTP Client event (TCPIP_TFTPC_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client and will be used by the DHCP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a TFTP Client event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the DHCP module.

Function

TCPIP_TFTPC_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_HANDLER handler,

const void* hParam)

TCPIP_TFTPC_SetServerAddress Function

Set the Server IP address for TFTP Client .

File

tftpc.h

C

```
void TCPIP_TFTPC_SetServerAddress(IP_MULTI_ADDRESS* ipAddr, IP_ADDRESS_TYPE ipType);
```

Returns

None

Description

This function is used to set the TFTP server address(either it will be IPv4 address or IPv6 address). This address will be used for either Get or Put mode of TFTP Client operation.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
ipAddr	pointer to the server address
ipType	type of address: IPv4/IPv6

Function

void TCPIP_TFTPC_SetServerAddress(IP_MULTI_ADDRESS* ipAddr,IP_ADDRESS_TYPE ipType)

TCPIP_TFTPC_SetCommand Function

TFTP client command operation configuration.

File

tftpc.h

C

```
TCPIP_TFTPC_OPERATION_RESULT TCPIP_TFTPC_SetCommand(IP_MULTI_ADDRESS* mAddr, IP_ADDRESS_TYPE ipType,
TCPIP_TFTP_CMD_TYPE cmdType, const char * fileName);
```

Returns

TFTPC_ERROR_BUSY - TFTP client is busy for one file processing and please retry later. when there is a TFTP operation going on, the another operation has to wait till the TFTP operation is completed. TFTPC_ERROR_INVALID_FILE_LENGTH - File Length should not be more than TCPIP_TFTPC_FILENAME_LEN. TFTPC_ERROR_NONE - Successful command operation.

Description

This function is used to set the client mode, server, and file name. The file name is accessed as per the TFTP command mode.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
mAddr	Server address
ipType	IP address type either IPv4 or IPv6 type
cmdType	GET or PUT command
fileName	File to be processed

Function

TCPIP_TFTPC_OPERATION_RESULT TCPIP_TFTPC_SetCommand(IP_MULTI_ADDRESS* mAddr,

- IP_ADDRESS_TYPE ipType,TCPIP_TFTP_CMD_TYPE cmdType,
- const char * fileName)

b) Data Types and Constants

TCPIP_TFTP_CMD_TYPE Enumeration

File command mode used for TFTP PUT and GET commands.

File

tftpc.h

C

```
typedef enum _TFTP_CMD_TYPE {
  TFTP_CMD_PUT_TYPE = 0,
  TFTP_CMD_GET_TYPE,
  TFTP_CMD_NONE
```

```
} TCPIP_TFTP_CMD_TYPE;
```

Members

Members	Description
TFTP_CMD_PUT_TYPE = 0	TFTP client issues a PUT command to write a file to the server
TFTP_CMD_GET_TYPE	TFTP client issues a GET command to read the file from the server

Description

Enumeration: TCPIP_TFTP_CMD_TYPE

These enum values are issued from the command line.

TCPIP_TFTPC_MODULE_CONFIG Structure

Placeholder for TFTP Client module configuration.

File

tftpc.h

C

```
typedef struct {
  const char* tftpc_interface;
  uint32_t tftpc_reply_timeout;
} TCPIP_TFTPC_MODULE_CONFIG;
```

Members

Members	Description
uint32_t tftpc_reply_timeout;	time-out for the server reply in seconds

Description

Structure: TCPIP_TFTPC_MODULE_CONFIG

This structure is a placeholder for TFTP Client module configuration.

TCPIP_TFTPC_EVENT_HANDLER Type

TFTPC event handler prototype.

File

tftpc.h

C

```
typedef void (* TCPIP_TFTPC_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_TYPE evType,void
*buf,uint32_t bufLen, const void* param);
```

Description

Type: TCPIP_TFTPC_EVENT_HANDLER

Prototype of a TFTPC event handler. Clients can register a handler with the TFTP service. Once an TFTP event occurs the TFTP Client service will be called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing! buf - Buffer is used to provide the memory Pointer . buf type need to be typecasted to char* while processing bufLen - The number of bytes present in the buffer.

TCPIP_TFTPC_EVENT_TYPE Enumeration

TFTP client Event Type

File

tftpc.h

C

```
typedef enum {
  TFTPC_EVENT_NONE = 0,
  TFTPC_EVENT_PUT_REQUEST = 0x0001,
  TFTPC_EVENT_GET_REQUEST = 0x0002,
```

```
  TFTPC_EVENT_ACKED = 0x0004,
  TFTP_EVENT_DATA_RECEIVED = 0x0008,
  TFTPC_EVENT_DECLINE = 0x0010,
  TFTPC_EVENT_TIMEOUT = 0x0020,
  TFTPC_EVENT_COMPLETED = 0x0040,
  TFTPC_EVENT_CONN_LOST = 0x0080,
  TFTPC_EVENT_CONN_ESTABLISHED = 0x0100,
  TFTPC_EVENT_SERVICE_DISABLED = 0x0200,
  TFTPC_EVENT_BUSY = 0x0400
} TCPIP_TFTPC_EVENT_TYPE;
```

Members

Members	Description
TFTPC_EVENT_NONE = 0	TFTP no event
TFTPC_EVENT_PUT_REQUEST = 0x0001	TFTP PUT request sent
TFTPC_EVENT_GET_REQUEST = 0x0002	TFTP GET Request sent
TFTPC_EVENT_ACKED = 0x0004	TFTP request acknowledge was received
TFTP_EVENT_DATA_RECEIVED = 0x0008	TFTP Client received Data for GET request
TFTPC_EVENT_DECLINE = 0x0010	TFTP File Put or Get communication declined due to Bad PDU
TFTPC_EVENT_TIMEOUT = 0x0020	TFTP server timeout
TFTPC_EVENT_COMPLETED = 0x0040	TFTP File Put or Get completed
TFTPC_EVENT_CONN_LOST = 0x0080	connection to the TFTP server lost
TFTPC_EVENT_CONN_ESTABLISHED = 0x0100	connection re-established
TFTPC_EVENT_SERVICE_DISABLED = 0x0200	TFTP service disabled
TFTPC_EVENT_BUSY = 0x0400	TFTP Client communication is going on

Description

Enumeration: TCPIP_TFTPC_EVENT_TYPE

None.

TCPIP_TFTPC_HANDLE Type

TFTPC handle.

File

tftpc.h

C

```
typedef const void* TCPIP_TFTPC_HANDLE;
```

Description

Type: TCPIP_TFTPC_HANDLE

A handle that a client can use after the event handler has been registered.

TCPIP_TFTPC_OPERATION_RESULT Enumeration

Standard error codes for TFTP PUT and GET command operation.

File

tftpc.h

C

```
typedef enum {
  TFTPC_ERROR_NONE = 0,
  TFTPC_ERROR_FILE_NOT_FOUND = -1,
  TFTPC_ERROR_BUSY = -2,
  TFTPC_ERROR_DISK_FULL = -3,
  TFTPC_ERROR_INVALID_OPERATION = -4,
  TFTPC_ERROR_UNKNOWN_TID = -5,
  TFTPC_ERROR_FILE_EXISTS = -6,
  TFTPC_ERROR_NO_SUCH_USE = -7,
  TFTPC_ERROR_DNS_RESOLVE_ERR = -8,
  TFTPC_ERROR_INVALID_INTERFACE = -9,
```

```
    TFTPC_ERROR_INVALID_FILE_LENGTH = -10,
    TFTPC_ERROR_INVALID_SERVER_ADDR = -11
  } TCPIP_TFTPC_OPERATION_RESULT;
```

Members

Members	Description
TFTPC_ERROR_FILE_NOT_FOUND = -1	TFTP client file not found
TFTPC_ERROR_BUSY = -2	TFTP client is busy when one file put or get transfer is going on.
TFTPC_ERROR_DISK_FULL = -3	TFTP client buffer full
TFTPC_ERROR_INVALID_OPERATION = -4	TFTP client invalid command operation
TFTPC_ERROR_UNKNOWN_TID = -5	TFTP ID error
TFTPC_ERROR_FILE_EXISTS = -6	TFTP client file already exists
TFTPC_ERROR_NO_SUCH_USE = -7	TFTP client not in use
TFTPC_ERROR_DNS_RESOLVE_ERR = -8	TFTP client DNS resolve error
TFTPC_ERROR_INVALID_INTERFACE = -9	TFTP client interface error
TFTPC_ERROR_INVALID_FILE_LENGTH = -10	TFTP client file length is more than the expected size, which should be the size of SYS_FS_MAX_PATH
TFTPC_ERROR_INVALID_SERVER_ADDR = -11	Invalid Server Address

Description

Enumeration: TCPIP_TFTPC_OPERATION_RESULT

This enumeration defines the standard error codes for TFTP PUT and GET command operation.

Files

Files

Name	Description
tftpc.h	The TFTP Client module implements the Trivial File Transfer Protocol (TFTP).
tftpc_config.h	TFTP Client configuration file.

Description

This section lists the source and header files used by the library.

tftpc.h

The TFTP Client module implements the Trivial File Transfer Protocol (TFTP).

Enumerations

	Name	Description
	_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTPC_EVENT_TYPE	TFTP client Event Type
	TCPIP_TFTPC_OPERATION_RESULT	Standard error codes for TFTP PUT and GET command operation.

Functions

	Name	Description
	TCPIP_TFTPC_GetEventNotification	Get the details of the TFTP client file mode communication event details.
	TCPIP_TFTPC_HandlerDeRegister	Deregisters a previously registered TFTP Client handler.
	TCPIP_TFTPC_HandlerRegister	Registers a TFTPC Handler.
	TCPIP_TFTPC_SetCommand	TFTP client command operation configuration.
	TCPIP_TFTPC_SetServerAddress	Set the Server IP address for TFTP Client .
	TCPIP_TFTPC_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_TFTPC_MODULE_CONFIG	Placeholder for TFTP Client module configuration.

Types

	Name	Description
	TCPIP_TFTPC_EVENT_HANDLER	TFTPC event handler prototype.
	TCPIP_TFTPC_HANDLE	TFTPC handle.

Description

TFTP Client Module API Header File

The TFTP Client module implements the Trivial File Transfer Protocol (TFTP). By default, the module opens a client socket for the default interface configuration. From the command prompt, the TFTP client module mode will be selected. At present only two modes are supported: Read and Write.

- For Read mode - File will be fetched from the Server using the GET command.
- For Write mode - Server will be able to fetch the file from the client using the PUT command.

The TFTP module needs the file system for GET and PUT command operation. When one mode is in operation, access to the other mode or another server is not allowed.

File Name

tftpc.h

Company

Microchip Technology Inc.

tftpc_config.h

TFTP Client configuration file.

Macros

	Name	Description
	TCPIP_TFTPC_ARP_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on PUT.
	TCPIP_TFTPC_CMD_PROCESS_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on GET.
	TCPIP_TFTPC_DEFAULT_IF	The default TFTP interface for multi-homed hosts.
	TCPIP_TFTPC_FILENAME_LEN	The maximum value for the file name size.
	TCPIP_TFTPC_HOSTNAME_LEN	The maximum TFTP host name length size.
	TCPIP_TFTPC_MAX_RETRIES	The number of attempts before declaring a TIMEOUT error.
	TCPIP_TFTPC_TASK_TICK_RATE	The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TFTPC_USER_NOTIFICATION	allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Description

Trivial File Transfer Protocol (TFTP) Client Configuration file

This file contains the TFTP Client module configuration options.

File Name

tftpc_config.h

Company

Microchip Technology Inc.

Telnet Module

This section describes the TCP/IP Stack Library Telnet module.

Introduction

TCP/IP Stack Library Telnet Module for Microchip Microcontrollers

This library provides the API of the Telnet (server) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Telnet, which runs over a TCP connection, provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network. The Telnet code included with Microchip's TCP/IP stack is a demonstration of the possible utilization of a Telnet server module. The server listens for a Telnet connection. When a client attempts to make one, the demonstration will prompt the client for a username and password, and if the correct one is provided, it will connect the client to a console service running in the system (if this console service is enabled). From this moment on, the Telnet connection can be used for various system and TCP/IP related commands, just as the regular system console is used.

The commands currently implemented and processed by the command processor can be dynamically updated, new commands can be added, etc. See the `tcpip_commands.c` file for the list of TCP/IP-related commands.

Based on this skeleton Telnet code, completely new behavior can be implemented for the Telnet server.

Using the Library

This topic describes the basic architecture of the Telnet TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `telnet.h`

The interface to the Telnet TCP/IP Stack library is defined in the `telnet.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Telnet TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

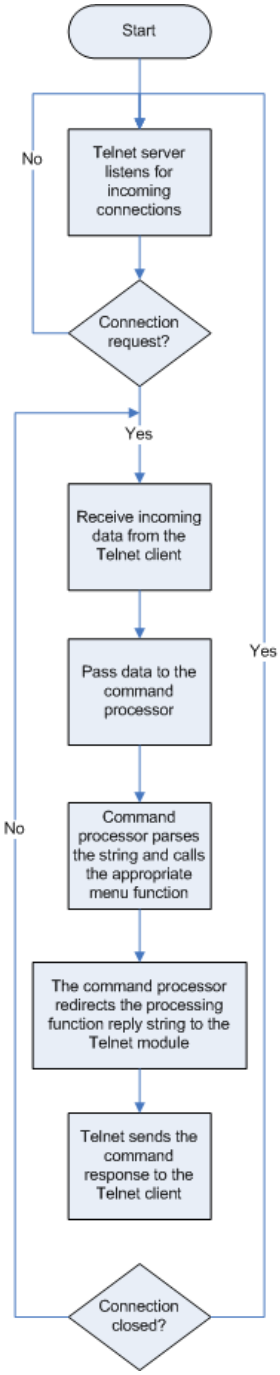
This library provides the API of the Telnet TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Currently, the Telnet TCP/IP Library runs in the background as a server and does not expose any API functions.

It serves incoming connections by sending the received data string to the command processor, which outputs each command reply to the Telnet channel.

Telnet 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 Telnet module.

Library Interface Section	Description
Functions	This section provides a routine that performs Telnet module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_TELNET_MAX_CONNECTIONS	Maximum number of Telnet connections
	TCPIP_TELNET_PASSWORD	Default Telnet password
	TCPIP_TELNET_TASK_TICK_RATE	telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TELNET_USERNAME	Default Telnet user name

Description

The configuration of the Telnet TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Telnet TCP/IP Stack Library. Based on the selections made, the Telnet TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Telnet TCP/IP Stack 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.

TCPIP_TELNET_MAX_CONNECTIONS Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_MAX_CONNECTIONS (2u)
```

Description

Maximum number of Telnet connections

TCPIP_TELNET_PASSWORD Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_PASSWORD "microchip"
```

Description

Default Telnet password

TCPIP_TELNET_TASK_TICK_RATE Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_TASK_TICK_RATE 100
```

Description

telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TELNET_USERNAME Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_USERNAME "admin"
```

Description

Default Telnet user name

Building the Library

This section lists the files that are available in the Telnet module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/telnet.c`	Telnet 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 Telnet module depends on the following modules:

- TCP/IP Stack Library
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_TELNET_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

Description

This section describes the Application Programming Interface (API) functions of the Telnet module.

Refer to each section for a detailed description.

a) Functions

TCPIP_TELNET_Task Function

Standard TCP/IP stack module task function.

File

telnet.h

C

```
void TCPIP_TELNET_Task();
```

Returns

None.

Description

This function performs Telnet module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The Telnet module should have been initialized.

Function

void TCPIP_TELNET_Task(void)

b) Data Types and Constants

TCPIP_TELNET_MODULE_CONFIG Structure

File

telnet.h

C

```
typedef struct {
} TCPIP_TELNET_MODULE_CONFIG;
```

Description

Telnet Configuration structure placeholder

Files

Files

Name	Description
telnet.h	Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network.
telnet_config.h	Configuration file

Description

This section lists the source and header files used by the library.

telnet.h

Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network.

Functions

	Name	Description
	TCPIP_TELNET_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

Description

Telnet Server Module for Microchip TCP/IP Stack. The telnet protocol is explained in RFC 854.

File Name

telnet.h

Company

Microchip Technology Inc.

telnet_config.h

Configuration file

Macros

	Name	Description
	TCPIP_TELNET_MAX_CONNECTIONS	Maximum number of Telnet connections
	TCPIP_TELNET_PASSWORD	Default Telnet password
	TCPIP_TELNET_TASK_TICK_RATE	telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TELNET_USERNAME	Default Telnet user name

Description

Telnet Configuration file

This file contains the Telnet module configuration options

File Name

telnet_config.h

Company

Microchip Technology Inc.

UDP Module

This section describes the TCP/IP Stack Library UDP module.

Introduction

TCP/IP Stack Library UDP Module for Microchip Microcontrollers

This library provides the API of the UDP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards.

Using the Library

This topic describes the basic architecture of the UDP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `udp.h`

The interface to the UDP TCP/IP Stack library is defined in the udp.h header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the UDP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

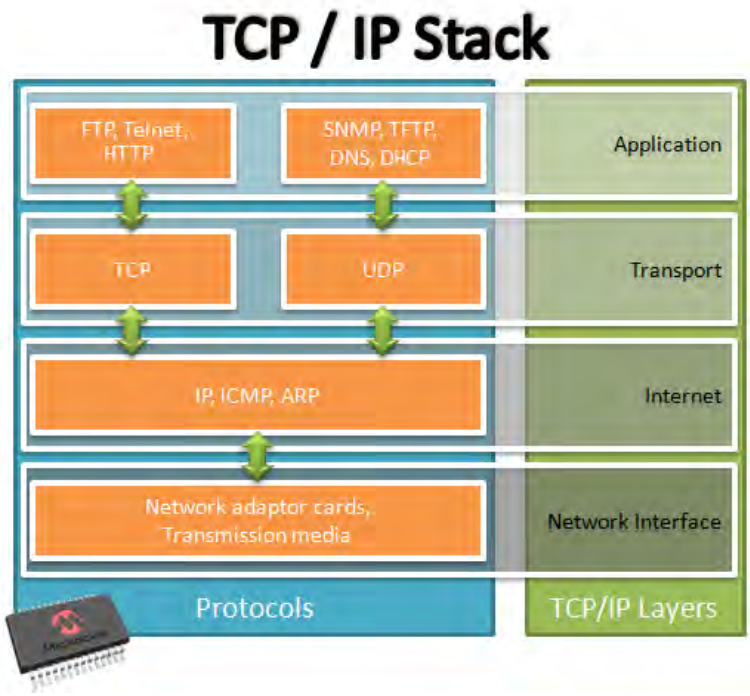
This library provides the API of the UDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

UDP Software Abstraction Block Diagram

This module provides software abstraction of the UDP module existent in any TCP/IP Stack implementation. It allows a user to UDP network traffic by opening and using UDP sockets.

Typical UDP Implementation



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the UDP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing UDP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions
Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

How the Library Works

This topic describes how the UDP TCP/IP Stack Library works.

Description

Connections over UDP should be thought of as data-gram based transfers. Each packet is a separate entity, the application should expect some packets to arrive out-of-order or even fail to reach the destination node. This is in contrast to TCP, in which the connection is thought of as a stream and network errors are automatically corrected. These tradeoffs in reliability are made for an increase in throughput. In general, UDP transfers operate two to three times faster than those made over TCP.

UDP sockets have their own TX buffers. Once the transmit buffer has valid data it is the socket user's responsibility to send the data over the network , calling the corresponding TCPIP_UDP_Flush function.

There is no state machine within the UDP module to automatically take care of the data transmission.

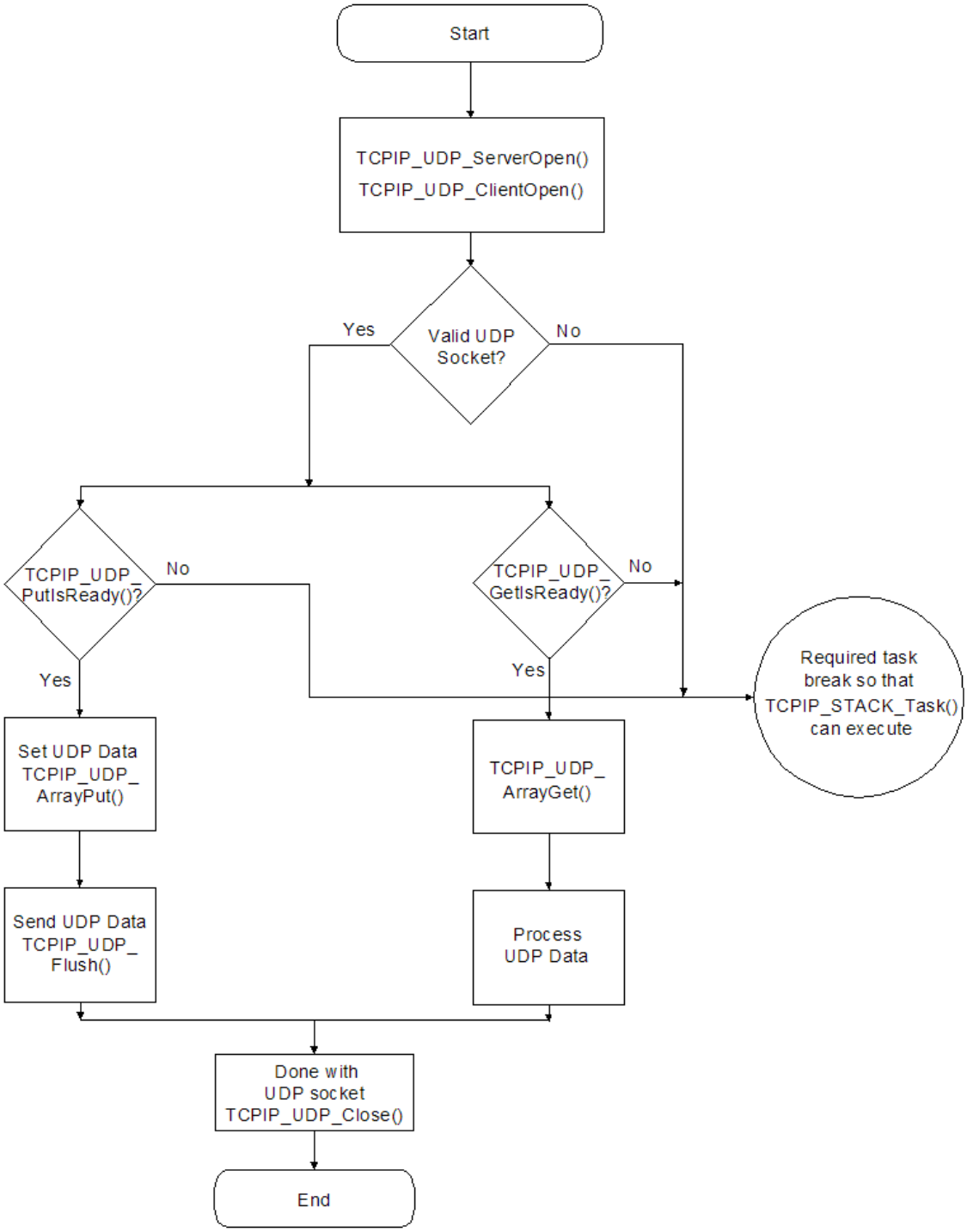
On the receive side, a UDP socket does not have its own RX buffer; however, it will use the RX buffer that was allocated by the corresponding MAC driver for receiving a network packet. What this means is that the user of the UDP socket will have to consume that pending RX data (or discard it), because eventually, the stack will run out of memory for other incoming traffic.

Core Functionality

This topic describes the core functionality of the UDP TCP/IP Stack Library.

Description

The following flow diagram provides an overview for the use of the UDP module:



Server/listening sockets are opened using TCPIP_UDP_ServerOpen. Client sockets are opened using TCPIP_UDP_ClientOpen. A client socket needs to have a remote address specified before it can transmit data.

Once the socket is opened, you can immediately begin transmitting data. To transmit a segment, call TCPIP_UDP_PutIsReady to determine how many bytes can be written. Then, use any of the TCPIP_UDP_ArrayPut family of functions to write data to the socket. Once all data has been written, call TCPIP_UDP_Flush to build and transmit the packet. Since each UDP socket has its own TX buffer, the sequence previously described can be executed in multiple steps. That data that is written in the socket is persistent and it will be stored until TCPIP_UDP_Flush is called.

Configuring the Library

Macros

	Name	Description
	TCPIP_UDP_MAX_SOCKETS	Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.
	TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT	The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the... more
	TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.
	TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).
	TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE	Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).
	TCPIP_UDP_SOCKET_POOL_BUFFERS	Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.
	TCPIP_UDP_USE_RX_CHECKSUM	Check incoming packets to have proper checksum.
	TCPIP_UDP_USE_TX_CHECKSUM	Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.
	TCPIP_UDP_USE_POOL_BUFFERS	enable the build of the pre-allocated pool buffers option

Description

The configuration of the UDP TCP/IP Stack Library is based on the file `system_config.h`. (which may include `udp_config.h`).

This header file contains the configuration selection for the UDP TCP/IP Stack Library. Based on the selections made, the UDP TCP/IP Stack Library may support the selected features. These configuration settings will apply to the single instance of the UDP TCP/IP Stack Library for all sockets.

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.

TCPIP_UDP_MAX_SOCKETS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_MAX_SOCKETS (10)
```

Description

Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.

TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT 3
```

Description

The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the amount of memory available to the stack.

TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT 3
```

Description

The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.

TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).

TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE 512
```

Description

Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).

TCPIP_UDP_SOCKET_POOL_BUFFERS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_POOL_BUFFERS 4
```

Description

Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.

TCPIP_UDP_USE_RX_CHECKSUM Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_RX_CHECKSUM
```

Description

Check incoming packets to have proper checksum.

TCPIP_UDP_USE_TX_CHECKSUM Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_TX_CHECKSUM
```

Description

Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.

TCPIP_UDP_USE_POOL_BUFFERS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_POOL_BUFFERS false
```

Description

enable the build of the pre-allocated pool buffers option

Building the Library

This section lists the files that are available in the UDP module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/udp.c`	UDP 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 UDP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Socket Management Functions

	Name	Description
	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
	TCPIP_UDP_IsOpened	Determines if a socket was opened.
	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
	TCPIP_UDP_Bind	Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.
	TCPIP_UDP_RemoteBind	Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.
	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
	TCPIP_UDP_OptionsGet	Allows getting the options for a socket such as current RX/TX buffer size, etc.
	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc
	TCPIP_UDP_SocketInfoGet	Returns information about a selected UDP socket.
	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket
	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket
	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.
	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket
	TCPIP_UDP_BcastIPV4AddressSet	Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.
	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket
	TCPIP_UDP_DestinationPortSet	Sets the destination port of a socket
	TCPIP_UDP_Disconnect	Disconnects a UDP socket and re-initializes it.
	TCPIP_UDP_SignalHandlerDeregister	Deregisters a previously registered UDP socket signal handler.
	TCPIP_UDP_SignalHandlerRegister	Registers a UDP socket signal handler.
	TCPIP_UDP_Task	Standard TCP/IP stack module task function.
	TCPIP_UDP_SocketsNumberGet	Returns the number of existent UDP sockets.

b) Transmit Data Transfer Functions

	Name	Description
	TCPIP_UDP_PutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket.

	TCPIP_UDP_StringPut	Writes a null-terminated string to the UDP socket.
	TCPIP_UDP_Put	Writes a byte to the UDP socket.
	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket.
	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket.

c) Receive Data Transfer Functions

	Name	Description
	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket.
	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket.
	TCPIP_UDP_Get	Reads a byte from the UDP socket.
	TCPIP_UDP_RxOffsetSet	Moves the read pointer within the socket RX buffer.
	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket.

d) Data Types and Constants

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	UDP_PORT	Defines a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE.
	UDP_SOCKET_INFO	Information about a socket
	UDP_SOCKET_OPTION	UDP socket options
	TCPIP_UDP_MODULE_CONFIG	UDP module run time configuration/initialization data.
	TCPIP_UDP_SIGNAL_FUNCTION	UDP Signal Handler.
	TCPIP_UDP_SIGNAL_HANDLE	UDP socket handle.
	TCPIP_UDP_SIGNAL_TYPE	UDP run-time signal types.

Description

This section describes the Application Programming Interface (API) functions of the UDP module.

Refer to each section for a detailed description.

a) Socket Management Functions

TCPIP_UDP_ServerOpen Function

Opens a UDP socket as a server.

File

udp.h

C

```
UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened.
- A UDP_SOCKET handle - Save this handle and use it when calling all other UDP APIs.

Description

Provides a unified method for opening UDP server sockets.

Preconditions

UDP is initialized.

Example

IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6. It can be IP_ADDRESS_TYPE_ANY and the server socket will accept any incoming type of connection.

UDP_PORT localPort - UDP port on which to listen for connections

IP_MULTI_ADDRESS* localAddress - Local address to use. Can be NULL if any incoming interface will do.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used.

Function

UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_UDP_ClientOpen Function

Opens a UDP socket as a client.

File

udp.h

C

```
UDP_SOCKET TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened.
- A UDP_SOCKET handle - Save this handle and use it when calling all other UDP APIs.

Description

Provides a unified method for opening UDP client sockets.

Remarks

None.

Preconditions

UDP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6. It can also be IP_ADDRESS_TYPE_ANY if the type of the socket will be specified later (using TCPIP_UDP_Bind, TCPIP_UDP_RemoteBind).
UDP_PORT remotePort	The remote UDP port to which a connection should be made. The local port for client sockets will be automatically picked by the UDP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to connect to. Not used if addType == IP_ADDRESS_TYPE_ANY. Can be NULL if later set by a call to TCPIP_UDP_RemoteBind.

Function

TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_UDP_IsOpened Macro

Determines if a socket was opened.

File

udp.h

C

```
#define TCPIP_UDP_IsOpened(hUDP) TCPIP_UDP_IsConnected(hUDP)
```

Description

This function determines if a socket was opened.

Remarks

This is a backward compatibility call.

Preconditions

None.

Parameters

Parameters	Description
hUDP	The socket to check.

Function

bool TCPIP_UDP_IsOpened(UDP_SOCKET hUDP)

TCPIP_UDP_IsConnected Function

Determines if a socket has an established connection.

File

udp.h

C

```
bool TCPIP_UDP_IsConnected(UDP_SOCKET hUDP);
```

Description

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen to determine when the connection is set up and ready for use.

Remarks

An UDP server or client socket can always send data, as long as it has a valid destination address, even if there's no remote socket connected to it. This function will return true if the socket is opened and currently has a valid destination. Note that this is true if a client socket was opened with a remote host address, a socket received data from a remote host, or a explicit remote bind was done.

Preconditions

None.

Parameters

Parameters	Description
hUDP	The socket to check.

Function

bool TCPIP_UDP_IsConnected(UDP_SOCKET hUDP)

TCPIP_UDP_Bind Function

Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.

File

udp.h

C

```
bool TCPIP_UDP_Bind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Sockets don't need specific binding, it is done automatically by the stack However, specific binding can be requested using this function. Works for

both client and server sockets. For a server socket it will restrict accepting connections of a specific type on a specific interface only. For a client socket it will force a specific address type and a local port and interface.

Remarks

If address type IP_ADDRESS_TYPE_ANY is used, the localAddress parameter won't be used and the socket won't change the local address. However, for the call to succeed, socket must have been created with IP_ADDRESS_TYPE_ANY type and must not be currently bound to a IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 type.

If address type IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 is used, then the socket type will be changed accordingly, and the localAddress will be used, if provided. The socket will be bound to this type of address. For the call to succeed the socket must currently have the type IP_ADDRESS_TYPE_ANY or to match the addType parameter.

If localAddress is the valid address of a network interface, and addType != IP_ADDRESS_TYPE_ANY, then the call will enforce UDP_OPTION_STRICT_NET on the socket.

If localPort is 0, the stack will assign a unique local port (if the socket doesn't already have a unique local port)

In order to change dynamically the type of address, the socket must have been created with a IP_ADDRESS_TYPE_ANY type and not currently bound to a different address type. TCPIP_UDP_Disconnect could be issued to remove a previous IP type binding.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket to bind.
addType	The type of address being used. Valid values are IP_ADDRESS_TYPE_ANY/IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
localPort	The local port to bind to.
localAddress	Local address to use.

Function

bool TCPIP_UDP_Bind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT localPort,
IP_MULTI_ADDRESS* localAddress);

TCPIP_UDP_RemoteBind Function

Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.

File

udp.h

C

```
bool TCPIP_UDP_RemoteBind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Sockets don't need specific remote binding, they should accept connections on any incoming interface. Therefore, the binding is done automatically by the stack. However, specific remote binding can be requested using this function. For a server socket it can be used to restrict accepting connections from a specific remote host. For a client socket it will just change the default binding done when the socket was opened.

Remarks

If address type IP_ADDRESS_TYPE_ANY is used, the remoteAddress parameter won't be used and the socket won't change the remote destination address. However, for the call to succeed, socket must have been created with IP_ADDRESS_TYPE_ANY type and must not be currently bound to a IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 type.

If address type IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 is used, then the socket type will be changed accordingly, and the remoteAddress will be used, if provided. The socket will be bound to this type of address. For the call to succeed the socket must currently have the type IP_ADDRESS_TYPE_ANY or to match the addType parameter.

If the remoteAddress != 0 and addType != IP_ADDRESS_TYPE_ANY, the call will enforce UDP_OPTION_STRICT_ADDRESS on the socket.

The remote port is always changed, even if remotePort == 0. It will enforce UDP_OPTION_STRICT_PORT on the socket.

In order to change dynamically the type of address, the socket must have been created with a IP_ADDRESS_TYPE_ANY type and not currently bound to a different address type. TCPIP_UDP_Disconnect could be issued to remove a previous IP type binding.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket to bind.
addType	The type of address being used. Valid values are IP_ADDRESS_TYPE_ANY/IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
remotePort	The remote port to bind to.
remoteAddress	Remote address to use.

Function

bool TCPIP_UDP_RemoteBind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
UDP_PORT remotePort, IP_MULTI_ADDRESS* remoteAddress);

TCPIP_UDP_Close Function

Closes a UDP socket and frees the handle.

File

udp.h

C

```
bool TCPIP_UDP_Close(UDP_SOCKET hUDP);
```

Returns

- true - If the call succeeded
- false - If the call failed (no such socket)

Description

Closes a UDP socket and frees the handle. Call this function to release a socket and return it to the pool for use by future communications.

Remarks

Always close the socket when no longer in use. This will free the allocated resources, including the TX buffers.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket handle to be released.

Function

bool TCPIP_UDP_Close(UDP_SOCKET hUDP)

TCPIP_UDP_OptionsGet Function

Allows getting the options for a socket such as current RX/TX buffer size, etc.

File

udp.h

C

```
bool TCPIP_UDP_OptionsGet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be retrieved at the socket level. This function provides compatibility with BSD implementations.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent
the size of the area has to be large enough to accommodate the specific option	• UDP_OPTION_STRICT_PORT - pointer to boolean • UDP_OPTION_STRICT_NET - pointer to boolean • UDP_OPTION_STRICT_ADDRESS - pointer to boolean • UDP_OPTION_BROADCAST - pointer to UDP_SOCKET_BCAST_TYPE • UDP_OPTION_BUFFER_POOL - pointer to boolean • UDP_OPTION_TX_BUFF - pointer to a 16 bit value to receive bytes of the TX buffer • UDP_OPTION_TX_QUEUE_LIMIT - pointer to an 8 bit value to receive the TX queue limit • UDP_OPTION_RX_QUEUE_LIMIT - pointer to an 8 bit value to receive the RX queue limit • UDP_OPTION_RX_AUTO_ADVANCE - pointer to boolean

Function

bool TCPIP_UDP_OptionsGet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);

TCPIP_UDP_OptionsSet Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

udp.h

C

```
bool TCPIP_UDP_OptionsSet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Remarks

Changing the UDP_OPTION_BUFFER_POOL will discard the data in the current socket buffer

- UDP_OPTION_TX_BUFF - 16-bit value in bytes of the TX buffer

the UDP_OPTION_TX_BUFF will discard the data in the current socket buffer

- UDP_OPTION_TX_QUEUE_LIMIT - 8-bit value of the TX queue limit
- UDP_OPTION_RX_QUEUE_LIMIT - 8-bit value of the RX queue limit
- UDP_OPTION_RX_AUTO_ADVANCE - boolean enable/disable

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket to set options for
option	specific option to be set

optParam	the option value; this is option dependent: • UDP_OPTION_STRICT_PORT - boolean enable/disable • UDP_OPTION_STRICT_NET - boolean enable/disable • UDP_OPTION_STRICT_ADDRESS - boolean enable/disable • UDP_OPTION_BROADCAST - UDP_SOCKET_BCAST_TYPE • UDP_OPTION_BUFFER_POOL - boolean enable/disable

Function

bool TCPIP_UDP_OptionsSet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);

TCPIP_UDP_SocketInfoGet Function

Returns information about a selected UDP socket.

File

udp.h

C

```
bool TCPIP_UDP_SocketInfoGet(UDP_SOCKET hUDP, UDP_SOCKET_INFO* pInfo);
```

Returns

- true - if the call succeeded
- false - if no such socket or invalid pInfo

Description

This function will fill a user passed UDP_SOCKET_INFO structure with status of the selected socket.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket pInfo - valid address of a UDP_SOCKET_INFO structure

Parameters

Parameters	Description
UDP_SOCKET hUDP	Socket for which information is to be obtained
UDP_SOCKET_INFO* pInfo	pointer to UDP_SOCKET_INFO to receive socket information

Function

bool TCPIP_UDP_SocketInfoGet(UDP_SOCKET hUDP, UDP_SOCKET_INFO* pInfo)

TCPIP_UDP_SocketNetGet Function

Gets the network interface of an UDP socket

File

udp.h

C

```
TCPIP_NET_HANDLE TCPIP_UDP_SocketNetGet(UDP_SOCKET hUDP);
```

Returns

Handle of the interface that socket currently uses.

Description

This function returns the interface handle of an UDP socket

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The UDP socket

Function

TCPIP_NET_HANDLE TCPIP_UDP_SocketNetGet(UDP_SOCKET hUDP)

TCPIP_UDP_SocketNetSet Function

Sets the network interface for an UDP socket

File

udp.h

C

```
bool TCPIP_UDP_SocketNetSet(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function sets the network interface for an UDP socket

Remarks

A NULL hNet can be passed (0) so that the current socket network interface selection will be cleared

If the hNet != 0, it will enforce UDP_OPTION_STRICT_NET on the socket.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The UDP socket
hNet	interface handle

Function

bool TCPIP_UDP_SocketNetSet(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet)

TCPIP_UDP_TxOffsetSet Function

Moves the pointer within the TX buffer.

File

udp.h

C

```
bool TCPIP_UDP_TxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset, bool relative);
```

Returns

- true - if the offset was valid and the write pointer has been moved
- false - if the offset was not valid

Description

This function allows the write location within the TX buffer to be specified. Future calls to TCPIP_UDP_Put, TCPIP_UDP_ArrayPut, TCPIP_UDP_StringPut, etc will write data from the indicated location.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset in the UDP packet data payload to move the write pointer.
relative	if true, the wOffset is added to the current write pointer. else the wOffset is from the beginning of the UDP buffer

Function

bool TCPIP_UDP_TxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset, bool relative)

TCPIP_UDP_SourceIPAddressSet Function

Sets the source IP address of a socket

File

udp.h

C

```
bool TCPIP_UDP_SourceIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, IP_MULTI_ADDRESS* localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure:
  - invalid socket
  - invalid socket address type
  - unspecified localAddress

Description

This function sets the IP source address, which allows changing the source P address dynamically.

Remarks

The call will fail if localAddress is 0. The source IP address will not be changed.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket localAddress - valid address pointer

Parameters

Parameters	Description
hUDP	the UDP socket
addType	Type of address: IPv4/IPv6
localAddress	pointer to an address to use

Function

bool TCPIP_UDP_SourceIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
IP_MULTI_ADDRESS* localAddress)

TCPIP_UDP_BcastIPV4AddressSet Function

Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.

File

udp.h

C

```
bool TCPIP_UDP_BcastIPV4AddressSet(UDP_SOCKET hUDP, UDP_SOCKET_BCAST_TYPE bcastType, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure:

- invalid socket
- invalid socket address type
- a broadcast for the specified interface could not be obtained
- invalid broadcast type specified

Description

It sets the broadcast address for the socket

Remarks

This function allows changing of the destination IPv4 address dynamically.

However, the call will fail if the socket was previously set to broadcast using the TCPIP_UDP_OptionsSet call. TCPIP_UDP_OptionsSet takes precedence.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	the UDP socket
bcastType	Type of broadcast
hNet	handle of an interface to use for the network directed broadcast Not used for network limited broadcast

Function

bool TCPIP_UDP_BcastIPV4AddressSet(UDP_SOCKET hUDP, UDP_SOCKET_BCAST_TYPE bcastType,
TCPIP_NET_HANDLE hNet)

TCPIP_UDP_DestinationIPAddressSet Function

Sets the destination IP address of a socket

File

udp.h

C

```
bool TCPIP_UDP_DestinationIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure:
  - invalid socket
  - invalid socket address type
  - socket is of broadcast type

Description

- It sets the IP destination address This allows changing the IP destination address dynamically.

Remarks

The call will fail if the socket was previously set to broadcast using the TCPIP_UDP_OptionsSet call. TCPIP_UDP_OptionsSet takes precedence.

The call will fail if remoteAddress is 0. The destination IP address will not be changed.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket remoteAddress - valid address pointer

Parameters

Parameters	Description
hUDP	the UDP socket
addType	Type of address: IPv4/IPv6
remoteAddress	pointer to an address to use

Function

bool TCPIP_UDP_DestinationIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_UDP_DestinationPortSet Function

Sets the destination port of a socket

File

udp.h

C

```
bool TCPIP_UDP_DestinationPortSet(UDP_SOCKET s, UDP_PORT remotePort);
```

Returns

- true - Indicates success
- false - Indicates an invalid socket

Description

This function sets the destination port, which allows changing the destination port dynamically.

Remarks

The destination remote port will always be changed, even if remotePort == 0.

It will not change the UDP_OPTION_STRICT_PORT on the socket.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	the UDP socket
remotePort	destination port to use

Function

bool TCPIP_UDP_DestinationPortSet(UDP_SOCKET s, UDP_PORT remotePort)

TCPIP_UDP_Disconnect Function

Disconnects a UDP socket and re-initializes it.

File

udp.h

C

```
bool TCPIP_UDP_Disconnect(UDP_SOCKET hUDP, bool flushRxQueue);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Disconnects a UDP socket and re-initializes it. Call this function to return the socket to its initial open state and to use it for future communication.

This function is meant especially for server sockets that could listen on multiple interfaces and on both IPv4 and IPv6 networks.

When a server socket received an inbound IPv4 connection it will be bound to IPv4 connections until it's closed or disconnected. Same is true for IPv6 connections.

Remarks

The call will try to maintain as much as possible from the socket state.

This will free the allocated TX buffers if the socket was opened with IP_ADDRESS_TYPE_ANY.

All the pending RX packets will be cleared when flushRxQueue is set. Otherwise the packets will be kept and will be available for next read

operations.

Note that if the RX pending packets exist and are not flushed, the socket will be bound immediately to the connection corresponding to the pending packet in the queue.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket handle to be disconnected.
flushRxQueue	boolean to flush the pending RX queue

Function

bool TCPIP_UDP_Disconnect(UDP_SOCKET hUDP, bool flushRxQueue)

TCPIP_UDP_SignalHandlerDeregister Function

Deregisters a previously registered UDP socket signal handler.

File

udp.h

C

```
bool TCPIP_UDP_SignalHandlerDeregister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_HANDLE hSig);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

Deregisters the UDP socket signal handler.

Preconditions

hSig valid UDP signal handle.

Parameters

Parameters	Description
s	The UDP socket
hSig	A handle returned by a previous call to TCPIP_UDP_SignalHandlerRegister.

Function

TCPIP_UDP_SignalHandlerDeregister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_HANDLE hSig)

TCPIP_UDP_SignalHandlerRegister Function

Registers a UDP socket signal handler.

File

udp.h

C

```
TCPIP_UDP_SIGNAL_HANDLE TCPIP_UDP_SignalHandlerRegister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_TYPE sigMask,
TCPIP_UDP_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (null handler, no such socket, existent handler).

Description

This function registers a UDP socket signal handler. The UDP module will call the registered handler when a UDP signal (TCPIP_UDP_SIGNAL_TYPE) occurs.

Remarks

Only one signal handler per socket is supported. A new handler does not override the existent one. Instead TCPIP_UDP_SignalHandlerDeregister has to be explicitly called.

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client but is currently not used and should be 0.

For multi-threaded systems the TCP/IP packet dispatch does not occur on the user thread. The signal handler will be called on a different thread context. It is essential that this handler is non blocking and really fast.

For multi-threaded systems, once set, it is not recommended to change the signal handler at run time. Synchronization between user threads and packet dispatch threads could be difficult. If really need to be changed, make sure that the old handler could still be called and it should be valid until the new one is taken into account. TCPIP_UDP_SignalHandlerDeregister needs to be called before registering another handler.

Preconditions

UDP valid socket.

Parameters

Parameters	Description
s	The UDP socket
sigMask	mask of signals to be reported
handler	signal handler to be called when a UDP signal occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the UDP module. Currently not used and it should be null.

Function

TCPIP_UDP_SignalHandlerRegister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_TYPE sigMask,
TCPIP_UDP_SIGNAL_FUNCTION handler, const void* hParam)

TCPIP_UDP_Task Function

Standard TCP/IP stack module task function.

File

udp.h

C

```
void TCPIP_UDP_Task();
```

Returns

None.

Description

This function performs UDP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The UDP module should have been initialized.

Function

void TCPIP_UDP_Task(void)

TCPIP_UDP_SocketsNumberGet Function

Returns the number of existent UDP sockets.

File

udp.h

C

```
int TCPIP_UDP_SocketsNumberGet();
```

Returns

The number of UDP sockets

Description

This function returns the number of created UDP sockets. This is the maximum number of sockets that can be opened at any moment as it's been passed as parameter when UDP module was created.

Preconditions

UDP module properly initialized

Function

int TCPIP_UDP_SocketsNumberGet(void)

b) Transmit Data Transfer Functions

TCPIP_UDP_PutIsReady Function

Determines how many bytes can be written to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_PutIsReady(UDP_SOCKET hUDP);
```

Returns

The number of bytes that can be written to this socket.

Description

This function determines how many bytes can be written to the specified UDP socket.

Remarks

If the current socket TX buffer is in use (in traffic), this function will allocate a new TX buffer. Otherwise the current TX buffer will be used.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_PutIsReady(UDP_SOCKET hUDP)

TCPIP_UDP_TxPutIsReady Function

Determines how many bytes can be written to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_TxPutIsReady(UDP_SOCKET hUDP, unsigned short count);
```

Returns

The number of bytes that can be written to this socket.

Description

This function returns the number of bytes that can be written to the specified UDP socket.

Remarks

The function won't increase the size of the UDP TX buffer. If this is needed use TCPIP_UDP_OptionsSet. The count variable is not used.

The function is similar to the TCPIP_UDP_PutIsReady and maintained for backward compatibility.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
count	Number of bytes requested

Function

uint16_t TCPIP_UDP_TxPutIsReady(UDP_SOCKET hUDP, unsigned short count)

TCPIP_UDP_ArrayPut Function

Writes an array of bytes to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_ArrayPut(UDP_SOCKET hUDP, const uint8_t * cData, uint16_t wDataLen);
```

Returns

The number of bytes successfully placed in the UDP transmit buffer. If this value is less than wDataLen, then the buffer became full and the input was truncated.

Description

This function writes an array of bytes to the UDP socket, while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket cData - valid pointer

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	The array to write to the socket.
wDataLen	Number of bytes from cData to be written.

Function

uint16_t TCPIP_UDP_ArrayPut(UDP_SOCKET hUDP, const uint8_t *cData, uint16_t wDataLen)

TCPIP_UDP_StringPut Function

Writes a null-terminated string to the UDP socket.

File

udp.h

C

```
const uint8_t* TCPIP_UDP_StringPut(UDP_SOCKET hUDP, const uint8_t * strData);
```

Returns

A pointer to the byte following the last byte written. Note that this is different than the TCPIP_UDP_ArrayPut functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

Description

This function writes a null-terminated string to the UDP socket while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket strData - valid pointer

Parameters

Parameters	Description
hUDP	UDP socket handle
strData	Pointer to the string to be written to the socket.

Function

uint8_t* TCPIP_UDP_StringPut(UDP_SOCKET hUDP, const uint8_t *strData)

TCPIP_UDP_Put Function

Writes a byte to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Put(UDP_SOCKET hUDP, uint8_t v);
```

Description

This function writes a single byte to the UDP socket, while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Remarks

This function is very inefficient and its use is discouraged. A buffered approach (TCPIP_UDP_ArrayPut) is preferred.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
v	The byte to be loaded into the transmit buffer.

Function

uint16_t TCPIP_UDP_Put(UDP_SOCKET hUDP, uint8_t v)

TCPIP_UDP_TxCountGet Function

Returns the amount of bytes written into the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_TxCountGet(UDP_SOCKET hUDP);
```

Description

This function returns the amount of bytes written into the UDP socket, i.e. the current position of the write pointer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_TxCountGet(UDP_SOCKET hUDP)

TCPIP_UDP_Flush Function

Transmits all pending data in a UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Flush(UDP_SOCKET hUDP);
```

Returns

The number of bytes that currently were in the socket TX buffer and have been flushed. Otherwise, 0 if the packet could not be transmitted:

- invalid socket
- invalid remote address
- no route to the remote host could be found

Description

This function builds a UDP packet with the pending TX data and marks it for transmission over the network interface. There is no UDP state machine to send the socket data automatically. The UDP socket client must call this function to actually send the data over the network.

Remarks

Note that a UDP socket must be flushed to send data over the network. There is no UDP state machine (auto transmit) for UDP sockets.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_Flush(UDP_SOCKET hUDP)

c) Receive Data Transfer Functions

TCPIP_UDP_GetIsReady Function

Determines how many bytes can be read from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_GetIsReady(UDP_SOCKET hUDP);
```

Returns

The number of bytes that can be read from this socket.

Description

This function will return the number of bytes that are available in the specified UDP socket RX buffer.

Remarks

The UDP socket queues incoming RX packets in an internal queue.

If currently there is no RX packet processed (as a result of retrieving all available bytes with TCPIP_UDP_ArrayGet, for example), this call will advance the RX packet to be processed to the next queued packet.

If a RX packet is currently processed, the call will return the number of bytes left to be read from this packet.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_GetIsReady(UDP_SOCKET hUDP)

TCPIP_UDP_ArrayGet Function

Reads an array of bytes from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_ArrayGet(UDP_SOCKET hUDP, uint8_t * cData, uint16_t wDataLen);
```

Returns

The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available.

Description

This function reads an array of bytes from the UDP socket, while adjusting the current read pointer and decrementing the remaining bytes available. TCPIP_UDP_GetIsReady should be used before calling this function to get the number of the available bytes in the socket.

Remarks

For discarding a number of bytes in the RX buffer the TCPIP_UDP_RxOffsetSet() can also be used.

The UDP socket queues incoming RX packets in an internal queue. This call will try to retrieve the bytes from the current processing packet but it won't advance the processed packet. TCPIP_UDP_GetIsReady should be called to advance the processed RX packet.

TCPIP_UDP_Discard should be called when done processing the current RX packet.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	The buffer to receive the bytes being read. If NULL, the bytes are simply discarded
wDataLen	Number of bytes to be read from the socket.

Function

uint16_t TCPIP_UDP_ArrayGet(UDP_SOCKET hUDP, uint8_t *cData, uint16_t wDataLen)

TCPIP_UDP_Get Function

Reads a byte from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Get(UDP_SOCKET hUDP, uint8_t * v);
```

Description

This function reads a single byte from the UDP socket, while decrementing the remaining RX buffer length. TCPIP_UDP_GetIsReady should be used before calling this function to get the number of bytes available in the socket.

Remarks

This function is very inefficient and its usage is discouraged. A buffered approach (TCPIP_UDP_ArrayGet) is preferred.

See the previous notes for TCPIP_UDP_ArrayGet function.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket handle
v	The buffer to receive the data being read.

Function

uint16_t TCPIP_UDP_Get(UDP_SOCKET hUDP, uint8_t *v)

TCPIP_UDP_RxOffsetSet Function

Moves the read pointer within the socket RX buffer.

File

udp.h

C

```
void TCPIP_UDP_RxOffsetSet(UDP_SOCKET hUDP, uint16_t rOffset);
```

Returns

None.

Description

This function allows the user to specify the read location within the socket RX buffer. Future calls to TCPIP_UDP_Get and TCPIP_UDP_ArrayGet will read data from the indicated location forward.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset from beginning of UDP packet data payload to place the read pointer.

Function

void TCPIP_UDP_RxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset)

TCPIP_UDP_Discard Function

Discards any remaining RX data from a UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Discard(UDP_SOCKET hUDP);
```

Returns

Number of discarded bytes, if any.

Description

This function discards any remaining received data in the UDP socket.

Remarks

The UDP socket queues incoming RX packets in an internal queue.

This call will discard the remaining bytes (if any) in the current RX packet and will advance the RX packet to be processed to the next queued packet.

This function should be normally called after retrieving the available bytes with TCPIP_UDP_ArrayGet.

When data available, calling it repeatedly will discard one pending RX packet at a time.

Note that a call to TCPIP_UDP_Discard is not needed if all bytes are retrieved with TCPIP_UDP_ArrayGet and then TCPIP_UDP_GetIsReady is called.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket handle

Function

uint16_t TCPIP_UDP_Discard(UDP_SOCKET hUDP)

d) Data Types and Constants

INVALID_UDP_SOCKET Macro

File

udp.h

C

```
#define INVALID_UDP_SOCKET (-1)        // Indicates a UDP socket that is not valid
```

Description

Indicates a UDP socket that is not valid

UDP_PORT Type

File

udp.h

C

```
typedef uint16_t UDP_PORT;
```

Description

Defines a UDP Port Number

UDP_SOCKET Type

File

udp.h

C

```
typedef int16_t UDP_SOCKET;
```

Description

Provides a handle to a UDP Socket

UDP_SOCKET_BCAST_TYPE Enumeration

File

udp.h

C

```
typedef enum {
  UDP_BCAST_NONE,
  UDP_BCAST_NETWORK_LIMITED,
  UDP_BCAST_NETWORK_DIRECTED
} UDP_SOCKET_BCAST_TYPE;
```

Members

Members	Description
UDP_BCAST_NONE	no broadcast
UDP_BCAST_NETWORK_LIMITED	network limited broadcast
UDP_BCAST_NETWORK_DIRECTED	network directed broadcast

Description

This is type UDP_SOCKET_BCAST_TYPE.

UDP_SOCKET_INFO Structure

File

udp.h

C

```
typedef struct {
  IP_ADDRESS_TYPE addressType;
  IP_MULTI_ADDRESS remoteIPaddress;
  IP_MULTI_ADDRESS localIPaddress;
  IP_MULTI_ADDRESS sourceIPaddress;
  IP_MULTI_ADDRESS destIPaddress;
  UDP_PORT remotePort;
  UDP_PORT localPort;
  TCPIP_NET_HANDLE hNet;
  uint16_t rxQueueSize;
  uint16_t txSize;
} UDP_SOCKET_INFO;
```

Members

Members	Description
IP_ADDRESS_TYPE addressType;	address type of the socket
IP_MULTI_ADDRESS remoteIPaddress;	current socket destination address
IP_MULTI_ADDRESS localIPaddress;	current socket source address
IP_MULTI_ADDRESS sourceIPaddress;	source address of the last packet
IP_MULTI_ADDRESS destIPaddress;	destination address of the last packet
UDP_PORT remotePort;	Port number associated with remote node
UDP_PORT localPort;	local port number
TCPIP_NET_HANDLE hNet;	associated interface
uint16_t rxQueueSize;	packets waiting in the rx queue
uint16_t txSize;	tx buffer size

Description

Information about a socket

UDP_SOCKET_OPTION Enumeration

File

udp.h

C

```
typedef enum {
  UDP_OPTION_STRICT_PORT,
  UDP_OPTION_STRICT_NET,
  UDP_OPTION_STRICT_ADDRESS,
  UDP_OPTION_BROADCAST,
  UDP_OPTION_BUFFER_POOL,
  UDP_OPTION_TX_BUFF,
  UDP_OPTION_TX_QUEUE_LIMIT,
  UDP_OPTION_RX_QUEUE_LIMIT,
  UDP_OPTION_RX_AUTO_ADVANCE
} UDP_SOCKET_OPTION;
```

Members

Members	Description
UDP_OPTION_STRICT_PORT	When connection is done the socket stores the remote host local port. If option is enabled the remote host local port is always checked to match the initial one. If disabled the remote host local port is not checked. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_STRICT_NET	When connection is done the socket stores the network interface the connection occurred on. If option is enabled the socket accepts data only from the interface that matches the initial connection. If disabled the socket receives data from any remote host regardless of the interface on which the packet arrived. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_STRICT_ADDRESS	When connection is done the socket stores the address of the remote host. If option is enabled the socket accepts data only from the host with address that matches the initial connection. If disabled the socket receives data from any remote host regardless of the address of that host. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_BROADCAST	Enables the Broadcast transmission by the socket
UDP_OPTION_BUFFER_POOL	Enables the socket to use the private UDP buffers pool. The size of the TX buffer has to be less than the size of the buffers in the pool.
UDP_OPTION_TX_BUFF	Request different TX buffer size. Has to call TCPIP_UDP_OptionsGet to see the exact space allocated.
UDP_OPTION_TX_QUEUE_LIMIT	Sets the maximum number of packets that can be queued/allocated by an UDP socket at a certain time. For sockets that need to transmit a lot of data (Iperf client, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the size of the UDP TX buffer, the connection speed and the amount of memory available to the stack.
UDP_OPTION_RX_QUEUE_LIMIT	Sets the maximum number of RX packets that can be queued by an UDP socket at a certain time. Setting this value to 0 will disable the socket receive functionality
UDP_OPTION_RX_AUTO_ADVANCE	Sets the RX auto advance option for a socket. The option is off by default If set, the option will automatically advance the socket to the next awaiting RX packet when a call to TCPIP_UDP_ArrayGet is made and no more data is available from the current packet. Setting this option forces the socket user to correctly monitor the number of bytes available and issue a TCPIP_UDP_Discard only when there's bytes left in the current packet. However sequential calls to TCPIP_UDP_ArrayGet can be made without the need to insert calls to TCPIP_UDP_Discard If cleared (default case) the socket user can safely issue a TCPIP_UDP_Discard even after calling TCPIP_UDP_ArrayGet to read all the bytes from the current RX packet Note that TCPIP_UDP_GetIsReady will always advance the current RX packet when no more data is available in the current packet. Even when the option is cleared a call to TCPIP_UDP_Discard is still not needed if all bytes are retrieved with TCPIP_UDP_ArrayGet and then TCPIP_UDP_GetIsReady is called.

Description

UDP socket options

TCPIP_UDP_MODULE_CONFIG Structure

UDP module run time configuration/initialization data.

File

udp.h

C

```
typedef struct {
  uint16_t nSockets;
  uint16_t sktTxBuffSize;
  uint16_t poolBuffers;
  uint16_t poolBufferSize;
} TCPIP_UDP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nSockets;	number of sockets to be created
uint16_t sktTxBuffSize;	default size of the socket TX buffer
uint16_t poolBuffers;	number of buffers in the pool; 0 if none
uint16_t poolBufferSize;	size of the buffers in the pool; all equal

Description

Structure: TCPIP_UDP_MODULE_CONFIG

UDP module configuration/initialization

TCPIP_UDP_SIGNAL_FUNCTION Type

UDP Signal Handler.

File

udp.h

C

```
typedef void (* TCPIP_UDP_SIGNAL_FUNCTION)(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet, TCPIP_UDP_SIGNAL_TYPE
sigType, const void* param);
```

Description

Type: TCPIP_UDP_SIGNAL_FUNCTION

Prototype of a UDP signal handler. Socket user can register a handler for the UDP socket. Once an UDP signals occurs the registered handler will be called.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

Parameters

Parameters	Description
hUDP	UDP socket to be used
hNet	the network interface on which the event has occurred
sigType	type of UDP signal that has occurred
param	additional parameter that can has been specified at the handler registration call Currently not used and it will be null.

TCPIP_UDP_SIGNAL_HANDLE Type

UDP socket handle.

File

udp.h

C

```
typedef const void* TCPIP_UDP_SIGNAL_HANDLE;
```

Description

Type: TCPIP_UDP_SIGNAL_HANDLE

A handle that a socket client can use after the signal handler has been registered.

TCPIP_UDP_SIGNAL_TYPE Enumeration

UDP run-time signal types.

File

udp.h

C

```
typedef enum {
  TCPIP_UDP_SIGNAL_TX_DONE = 0x0001,
  TCPIP_UDP_SIGNAL_RX_DATA = 0x0100
} TCPIP_UDP_SIGNAL_TYPE;
```

Members

Members	Description
TCPIP_UDP_SIGNAL_TX_DONE = 0x0001	A data packet was successfully transmitted on the interface There may be available buffer space to send new data
TCPIP_UDP_SIGNAL_RX_DATA = 0x0100	A data packet was successfully received and there is data available for this socket

Description

Enumeration: TCPIP_UDP_SIGNAL_TYPE

Description of the signals/events that a UDP socket can generate.

Remarks

These signals are used in the socket event handling notification functions. Currently a UDP notification doesn't set multiple flags as the TX and RX events are handled separately.

The signals are 16 bits wide.

Files

Files

Name	Description
udp.h	UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards
udp_config.h	UDP Configuration file

Description

This section lists the source and header files used by the library.

udp.h

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards

Enumerations

	Name	Description
	TCPIP_UDP_SIGNAL_TYPE	UDP run-time signal types.
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE.
	UDP_SOCKET_OPTION	UDP socket options

Functions

	Name	Description
	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket.
	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket.
	TCPIP_UDP_BcastIPV4AddressSet	Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.
	TCPIP_UDP_Bind	Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.
	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket
	TCPIP_UDP_DestinationPortSet	Sets the destination port of a socket
	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket.
	TCPIP_UDP_Disconnect	Disconnects a UDP socket and re-initializes it.
	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket.
	TCPIP_UDP_Get	Reads a byte from the UDP socket.
	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket.
	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
	TCPIP_UDP_OptionsGet	Allows getting the options for a socket such as current RX/TX buffer size, etc.
	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc
	TCPIP_UDP_Put	Writes a byte to the UDP socket.
	TCPIP_UDP_PutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_RemoteBind	Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.
	TCPIP_UDP_RxOffsetSet	Moves the read pointer within the socket RX buffer.
	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
	TCPIP_UDP_SignalHandlerDeregister	Deregisters a previously registered UDP socket signal handler.
	TCPIP_UDP_SignalHandlerRegister	Registers a UDP socket signal handler.
	TCPIP_UDP_SocketInfoGet	Returns information about a selected UDP socket.
	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket
	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket
	TCPIP_UDP_SocketsNumberGet	Returns the number of existent UDP sockets.
	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket
	TCPIP_UDP_StringPut	Writes a null-terminated string to the UDP socket.
	TCPIP_UDP_Task	Standard TCP/IP stack module task function.
	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket.
	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.
	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket.

Macros

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	TCPIP_UDP_IsOpened	Determines if a socket was opened.

Structures

	Name	Description
	TCPIP_UDP_MODULE_CONFIG	UDP module run time configuration/initialization data.
	UDP_SOCKET_INFO	Information about a socket

Types

	Name	Description
	TCPIP_UDP_SIGNAL_FUNCTION	UDP Signal Handler.
	TCPIP_UDP_SIGNAL_HANDLE	UDP socket handle.
	UDP_PORT	Defines a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket

Description

UDP Module Definitions for the Microchip TCP/IP Stack

UDP is a standard transport layer protocol described in RFC 768.

File Name

udp.h

Company

Microchip Technology Inc.

udp_config.h

UDP Configuration file

Macros

	Name	Description
	TCPIP_UDP_MAX_SOCKETS	Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.
	TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT	The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the... more
	TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.
	TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).
	TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE	Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).
	TCPIP_UDP_SOCKET_POOL_BUFFERS	Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.
	TCPIP_UDP_USE_POOL_BUFFERS	enable the build of the pre-allocated pool buffers option
	TCPIP_UDP_USE_RX_CHECKSUM	Check incoming packets to have proper checksum.
	TCPIP_UDP_USE_TX_CHECKSUM	Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.

Description

User Datagram Protocol (UDP) Configuration file

This file contains the UDP module configuration options

File Name

udp_config.h

Company

Microchip Technology Inc.

Zeroconf Module

This section describes the TCP/IP Stack Library Zeroconf module.

Introduction

TCP/IP Stack Library Zero Configuration (Zeroconf) Module for Microchip Microcontrollers

This library provides the API of the Zeroconf module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

Using the Library

This topic describes the basic architecture of the Zeroconf TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header Files: `tcpip_helper.h`, `zero_conf_link_local.h`, and `zero_conf_multicast_dns.h`

The interface to the Zeroconf TCP/IP Stack library is defined in the `tcpip_helper.h`, `zero_conf_link_local.h`, and `zero_conf_multicast_dns.h` header files. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Zeroconf TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Zeroconf TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Zeroconf Software Abstraction Model

Bonjour Service 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 Zeroconf module.

Library Interface Section	Description
Multicast DNS Functions	This section provides multicast DNS interface routines to Zeroconf
Link Local Functions	This section provides Link Local interface routines to Zeroconf

Data Types and Constants	This section provides various definitions describing this API

Core Functionality

This section provides information on core functionality.

Enabling

Zeroconf can be enabled in the MPLAB Harmony Configurator, or by setting the following two defines in `system_config.h` file:

- TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL
- TCPIP_STACK_USE_ZEROCONF_MDNS_SD

Link Local (Zeroconf)

The first component of Zeroconf is the ability to self-assign an IP address to each member of a network. Normally, a DHCP server would handle such situations. However, in cases where no DHCP server exists, Zeroconf enabled devices negotiate unique IP addresses amongst themselves.

mDNS

The second component of Zeroconf is the ability to self-assign human-readable hostnames for themselves. Multicast DNS provides a local network the ability to have the features of a DNS server. Users can use easily remembered hostnames to accesses the devices on the network. In the event that devices elect to use the same hostname, as in the IP address resolution, each of the devices will auto-negotiate new names for themselves (usually by appending a number to the end of the name).

Service Discovery

The last component of Zeroconf is service discovery. All Zeroconf devices can broadcast what services they provide. For instance, a printer can broadcast that it has printing services available. A thermostat can broadcast that it has an HVAC control service. Other interested parties on the network who are looking for certain services can then see a list of devices that have the capability of providing the service, and connect directly to it. This further eliminates the need to know whether something exists on a network (and what its IP or hostname is). As an end-user, all you would need to do is query the network if a certain service exists, and easily connect to it.

Demonstration

The demonstration, when enabled, shows all three items working together. Each development kit in the network assumes the hostname of MCHPBOARD-x.local, where 'x' is an number incrementing from 1 (only in the case where multiple kits are programmed for the network). Each board will broadcast its service, which is the DemoWebServer.

Zeroconf Enabled Environments

All Apple products have Zeroconf enabled by default. On Windows, you'll need to download the Safari web browser, and during the install, enable support for Bonjour. Note that in the Safari browser, you can browse and see a list of all Bonjour enabled devices, and click through to them automatically.

Configuring the Library

The configuration of the Zeroconf TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Zeroconf TCP/IP Stack Library. Based on the selections made, the Zeroconf TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Zeroconf TCP/IP Stack 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 Zeroconf module of the TCP/IP Stack 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/tcpip`.

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
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

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/zero_conf_helper.c`	Zeroconf helper function implementation file.
`/src/zero_conf_link_local.c`	Zeroconf local link configuration implementation file.
`/src/zero_conf_multicast_dns.c`	Service advertisement and Multicast DNS 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 Zeroconf module depends on the following modules:

- TCP/IP Stack Library
- ARP Module
- UDP Module

Library Interface

a) Multicast DNS Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.
	TCPIP_MDNS_ServiceRegister	DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.
	TCPIP_MDNS_Task	Standard TCP/IP stack module task function.

b) Link Local Functions

	Name	Description
	TCPIP_ZCLL_Disable	Disables Zero Configuration on the specified interface.
	TCPIP_ZCLL_Enable	Enables Zero Configuration on the specified interface.
	TCPIP_ZCLL_IsEnabled	Returns whether or not an interface is enabled for zero configuration
	TCPIP_ZCLL_Task	Standard TCP/IP stack module task function.

c) Data Types and Constants

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);
	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

Description

This section describes the Application Programming Interface (API) functions of the Zeroconf module.

Refer to each section for a detailed description.

a) Multicast DNS Functions

TCPIP_MDNS_ServiceDeregister Function

DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceDeregister(TCPIP_NET_HANDLE netH);
```

Returns

MDNSD_ERR_CODE - Returns an error code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - Returns on success of call
- MDNSD_ERR_INVAL - When the input parameters are invalid or if the function is invoked in an invalid state

Description

This function is used by end-user application to deregister DNS Service Discovery on a local network. When this gets invoked by end-user DNS SD stack sends out Good-Bye packets to update all peer machines that service will no longer be present. All peer machines remove the corresponding entry from the Browser list.

This is the last functions that needs to be invoked by the end-user application to free the DNS SD stack for some other application.

Preconditions

TCPIP_MDNS_ServiceRegister must be invoked before this call.

Parameters

Parameters	Description
netH	handle of the network to be deregistered

Function

MDNSD_ERR_CODE TCPIP_MDNS_ServiceDeregister()

TCPIP_MDNS_ServiceRegister Function

DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceRegister(TCPIP_NET_HANDLE netH, const char * srv_name, const char *
srv_type, uint16_t port, const uint8_t * txt_record, uint8_t auto_rename, void (*call_back)(char *name,
MDNSD_ERR_CODE err, void *context), void * context);
```

Returns

MDNSD_ERR_CODE - Returns Error-code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - returns on success of call
- MDNSD_ERR_BUSY - When already some other service is being advertised using this DNS SD stack
- MDNSD_ERR_INVAL - Invalid parameter

Description

This function is used by end-user application to announce its service on local network. All peer machines that are compliant with Multicast DNS and DNS Service Discovery protocol can detect the announcement and lists out an entry in the Service Browser list. The end-user selects an entry to connect to this service. So ultimately this is an aid to end-user to discover any service of interest on a local network.

This is the first function that needs to be invoked by end-user application. Presently Multicast-DNS and Service discovery stack supports only single service advertisement. Once the application wants to terminate the service it has to invoke TCPIP_MDNS_ServiceDeregister function to free the DNS SD stack for some other application.

Preconditions

None.

Parameters

Parameters	Description
netH	handle of the network to be registered
srv_name	Service Name, which is being advertised
srv_type	For a HTTP-Service its "_http._tcp.local"
_http	is application protocol proceeded with an underscore
_tcp	is lower-layer protocol on which service runs
local	is to represent service is on local-network For a iTunes Music Sharing "_daap._tcp.local" For a Printing Service "_ipp._tcp.local" Refer to http://www.dns-sd.org/ServiceTypes.html for more service types.
port	Port number on which service is running
txt_len	For additional information about service like default page (e.g., "index.htm") for HTTP-service. Length of such additional information
txt_record	String of additional information (e.g., "index.htm") for HTTP service.
auto_rename	A flag to indicate DNS-SD stack, whether to rename the service automatically or not. If this is set to '0' Callback parameter will be used to indicate the conflict error and user has to select different name and re-register with this function. If this is set to '1' service-name will be automatically renamed with numerical suffix.
callback	Callback function, which is user-application defined. This callback gets invoked on completion of service advertisement. If an service name-conflict error is detected and auto_rename is set to '0' callback gets invoked with MDNSD_ERR_CONFLICT as error-code.
context	Opaque context (pointer to opaque data), which needs to be used in callback function.

Function

MDNSD_ERR_CODE TCPIP_MDNS_ServiceRegister(...)

TCPIP_MDNS_ServiceUpdate Function

DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceUpdate(TCPIP_NET_HANDLE netH, uint16_t port, const uint8_t * txt_record);
```

Returns

MDNSD_ERR_CODE - Returns an error code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - returns on success of call
- MDNSD_ERR_INVAL - When the input parameters are invalid or if the function is invoked in invalid state

Description

This function is used by the end-user application to update its service, which was previously registered. With this function, the end-user application updates the port number on which the service is running. It can also update the additional information of service. For example, the default page can be updated to new page and corresponding page name can be input to this function to update all peer machines. The modified service will be announced with new contents on local network.

This is an optional function and should be invoked only if it is necessary.

Preconditions

TCPIP_MDNS_ServiceRegister must be invoked before this call.

Parameters

Parameters	Description
netH	Handle of the network to perform the service update
port	Port number on which service is running
txt_record	String of additional information (e.g., "index.htm") for HTTP-service.

Function

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceUpdate(
    TCPIP_NET_HANDLE netH,
uint16_t port,
const uint8_t *txt_record)
```

TCPIP_MDNS_Task Function

Standard TCP/IP stack module task function.

File

zero_conf_multicast_dns.h

C

```
void TCPIP_MDNS_Task();
```

Returns

None.

Description

This function performs MDNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

MDNS module should have been initialized.

Function

void TCPIP_MDNS_Task(void)

b) Link Local Functions

TCPIP_ZCLL_Disable Function

Disables Zero Configuration on the specified interface.

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This API is used by end-user application to disable Zero Configuration on a specific interface.

Remarks

None.

Preconditions

The TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be disabled

Function

bool TCPIP_ZCLL_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_Enable Function

Enables Zero Configuration on the specified interface.

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This API is used by end-user application to enable Zero Configuration on a specific interface.

Remarks

None.

Preconditions

TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be enabled

Function

bool TCPIP_ZCLL_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_IsEnabled Function

Returns whether or not an interface is enabled for zero configuration

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - interface is enabled for zero configuration
- false - interface is not enabled for zero configuration

Description

This API is used by end-user application check whether or not that an interface is enabled for zero configuration.

Preconditions

The TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be examined

Function

bool TCPIP_ZCLL_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_Task Function

Standard TCP/IP stack module task function.

File

zero_conf_link_local.h

C

```
void TCPIP_ZCLL_Task();
```

Returns

None.

Description

This function performs ZCLL module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ZCLL module should have been initialized.

Function

void TCPIP_ZCLL_Task(void)

c) Data Types and Constants

MDNSD_ERR_CODE Enumeration

File

zero_conf_multicast_dns.h

C

```
typedef enum {
  MDNSD_SUCCESS = 0,
  MDNSD_ERR_BUSY = 1,
  MDNSD_ERR_CONFLICT = 2,
  MDNSD_ERR_INVAL = 3
} MDNSD_ERR_CODE;
```

Members

Members	Description
MDNSD_ERR_BUSY = 1	Already Being used for another Service
MDNSD_ERR_CONFLICT = 2	Name Conflict
MDNSD_ERR_INVAL = 3	Invalid Parameter

Description

void DisplayHostName(uint8_t *HostName);

ZCLL_MODULE_CONFIG Structure

File

zero_conf_link_local.h

C

```
typedef struct {
} ZCLL_MODULE_CONFIG;
```

Description

Placeholder for Zero Configuration Link Layer module configuration.

Files

Files

Name	Description
zero_conf_link_local.h	Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.
zero_conf_multicast_dns.h	Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Description

This section lists the source and header files used by the library.

zero_conf_link_local.h

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Functions

	Name	Description
	TCPIP_ZCLL_Disable	Disables Zero Configuration on the specified interface.
	TCPIP_ZCLL_Enable	Enables Zero Configuration on the specified interface.
	TCPIP_ZCLL_IsEnabled	Returns whether or not an interface is enabled for zero configuration
	TCPIP_ZCLL_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

Description

Zero Configuration (Zeroconf) IPV4 Link Local Addressing Module for the Microchip TCP/IP Stack

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

File Name

zero_conf_link_local.h

Company

Microchip Technology Inc.

zero_conf_multicast_dns.h

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Enumerations

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);

Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.
	TCPIP_MDNS_ServiceRegister	DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.

	TCPIP_MDNS_Task	Standard TCP/IP stack module task function.

Description

Zero Configuration (Zeroconf) Multicast DNS and Service Discovery Module for Microchip TCP/IP Stack

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

File Name

zero_conf_multicast_dns.h

Company

Microchip Technology Inc.

Index

_

__ss_aligntype macro 83

_BERKELEY_API_HEADER_FILE macro 83

_IPV6_ADDR_STRUCT structure 487

_IPV6_DATA_SEGMENT_HEADER structure 370

_IPV6_PACKET structure 373

_IPV6_RX_FRAGMENT_BUFFER structure 375

_SS_PADSIZE macro 84

_SS_SIZE macro 84

_tag_MAC_DATA_SEGMENT structure 406

_tag_TCPIP_HTTP_NET_USER_CALLBACK structure 286

_tag_TCPIP_ICMP_ECHO_REQUEST structure 302

_tag_TCPIP_MAC_PACKET structure 412

_TFTP_CMD_TYPE enumeration 658

A

Abstraction Model 28, 32, 52, 88, 102, 118, 139, 162, 173, 182, 189, 228, 294, 305, 327, 342, 382, 432, 500, 504, 512, 515, 529, 550, 599, 610, 663, 669, 703

- Announce Module 28
- ARP Module 32
- Berkeley Module 52
- DHCP Module 88
- DHCP Server Module 102
- DHCPv6 Module 118
- DNS Module 139
- DNSS Module 162
- Dynamic DNS Module 173
- FTP Module 182
- HTTP Module 189
- HTTP Net Module 228
- ICMP Module 294
- ICMPv6 Module 305
- IPv4 Module 327
- IPv6 Module 342
- MAC Driver Module 382
- Manager Module 432
- NBNS Module 500
- NDP Module 504
- Reboot Module 512
- SMTP Module 515
- SMTPC Module 529
- SNMP Module 550
- SNTP Module 599
- TCP Module 610
- Telnet Module 663
- UDP Module 669
- Zeroconf Module 703

accept function 56

addrinfo structure 71

AF_INET macro 67

AF_INET6 macro 83

Announce Module 28

ARP Changes 11

ARP Module 32

arp.h 50

ARP_CACHE_DELETE_OLD macro 34

arp_config.h 50

Authentication 192

B

Berkeley Changes 11

Berkeley Module 52

berkeley_api.h 84

berkeley_api_config.h 87

BERKELEY_MODULE_CONFIG structure 84

bind function 57

Building the Library 6, 30, 36, 54, 89, 105, 126, 143, 165, 174, 185, 199, 242, 296, 306, 325, 331, 347, 390, 436, 501, 509, 513, 519, 534, 556, 603, 617, 654, 666, 674, 704

- Announce Module 30
- ARP Module 36
- Berkeley Module 54
- DHCP Module 89
- DHCP Server Module 105
- DHCPv6 Module 126
- DNS Module 143
- DNSS Module 165
- Dynamic DNS Module 174
- FTP Module 185
- HTTP Module 199
- HTTP Net Module 242
- ICMP Module 296
- ICMPv6 Module 306
- IPv4 Module 331
- IPv6 Module 347
- MAC Driver Module 390
- Manager Module 436
- NBNS Module 501
- NDP Module 509
- Reboot Module 513
- SMTP Module 519
- SMTPC Module 534
- SNMP Module 556
- SNTP Module 603
- TCP Module 617
- TCPIP Stack Library 6
- Telnet Module 666
- TFTP Module 654
- UDP Module 674
- Zeroconf Module 704

C

closesocket function 57

Compression 194

Configuring the Library 29, 33, 53, 89, 103, 119, 140, 163, 174, 183, 194, 232, 295, 306, 330, 343, 386, 436, 500, 506, 512, 518, 530, 551, 600, 612, 652, 665, 672, 704

- Announce Module 29
- ARP Module 33
- Berkeley Module 53
- DHCP Module 89
- DHCP Server Module 103
- DHCPv6 Module 119
- DNS Module 140

DNSS Module 163
Dynamic DNS Module 174
FTP Module 183
HTTP Module 194
HTTP Net Module 232
ICMP Module 295
ICMPv6 Module 306
IPv4 Module 330
IPv6 Module 343
MAC Driver Module 386
Manager Module 436
NBNS Module 500
NDP Module 506
Reboot Module 512
SMPT Module 518, 530
SNMP Module 551
SNTP Module 600
TCP Module 612
Telnet Module 665
TFTP Module 652
UDP Module 672
Zeroconf Module 704
Configuring the Module 323
Iperf Module 323
connect function 58
Cookies 193
Core Functionality 328, 384, 434, 611, 670, 704

D

ddns.h 180
ddns_config.h 180
DDNS_MODULE_CONFIG structure 177
DDNS_POINTERS structure 177
DDNS_SERVICES enumeration 178
DDNS_STATUS enumeration 179
Demonstration 704
Development Information (Advance Information) 23
DHCP Module 88
DHCP Server Module 102
dhcp.h 100
dhcp_config.h 101
dhcps.h 115
dhcps_config.h 116
DHCPv6 Module 118
dhcpv6.h 135
dhcpv6_config.h 136
Discovering the Board 28
DNS Module 138
DNS Server Module 161
dns.h 159
dns_config.h 160
dnss.h 171
dnss_config.h 172
Dynamic DNS Module 173
Dynamic Variables 190
Dynamic Variables Processing 230

E

EAI_AGAIN macro 71
EAI_BADFLAGS macro 72
EAI_FAIL macro 72
EAI_FAMILY macro 72
EAI_MEMORY macro 72
EAI_NONAME macro 72
EAI_OVERFLOW macro 73
EAI_SERVICE macro 73
EAI_SOCKTYPE macro 73
EAI_SYSTEM macro 73
Enabling 704

F

File Processing 230
Files 30, 49, 84, 100, 115, 135, 158, 171, 180, 187, 225, 289, 303, 315, 325, 340, 378, 428, 496, 503, 510, 514, 527, 547, 594, 608, 646, 661, 667, 699, 711
Announce Module 30
ARP Module 49
Berkeley Module 84
DHCP Module 100, 115
DHCPv6 Module 135
DNS Module 158
DNSS Module 171
Dynamic DNS Module 180
FTP Module 187
HTTP Module 225
HTTP Net Module 289
ICMP Module 303
ICMPv6 Module 315
Iperf Module 325
IPv4 Module 340
IPv6 Module 378
MAC Driver Module 428
Manager Module 496
NBNS Module 503
NDP Module 510
Reboot Module 514
SMTP Module 527
SMTPC Module 547
SNMP Module 594
SNTP Module 608
TCP Module 646
Telnet Module 667
TFTP Module 661
UDP Module 699
Zeroconf Module 711
Flash and RAM Usage 4
Form Processing 192
freeaddrinfo function 66
FTP Module 182
ftp.h 187
ftp_config.h 187

G

getaddrinfo function 66
gethostbyname function 64

gethostname function 59
getsockname function 65
getsockopt function 63
Getting Help 6

H

HOST_NOT_FOUND macro 81
hostent structure 80
How the Library Works 33, 89, 103, 119, 328, 383, 433, 611, 651, 670
    ARP Module 33
    DHCP Module 89
    DHCP Server Module 103
    DHCPv6 Module 119
    IPv4 Module 328
    MAC Driver Module 383
    Manager Module 433
    TCP Module 611
    UDP Module 670
HTTP Changes 12
HTTP Configuration 560
HTTP Features 190
HTTP Module 189
HTTP Net Features 229
HTTP Net Module 228
http.h 225
http_config.h 226
HTTP_CONN_HANDLE type 222
HTTP_FILE_TYPE enumeration 222
HTTP_IO_RESULT enumeration 222
HTTP_MODULE_FLAGS enumeration 223
http_net.h 289
http_net_config.h 291
HTTP_READ_STATUS enumeration 223
HTTP_STATUS enumeration 223

I

ICMP Module 294
icmp.h 303
icmp_config.h 304
ICMP_ECHO_RESULT enumeration 301
ICMP_HANDLE type 302
ICMP6_FILTER macro 73
ICMPv6 Module 305
icmpv6.h 315
icmpv6_config.h 316
ICMPV6_ERR_DU_CODE enumeration 312
ICMPV6_ERR_PP_CODE enumeration 312
ICMPV6_ERR_PTB_CODE macro 310
ICMPV6_ERR_TE_CODE enumeration 313
ICMPV6_HANDLE type 313
ICMPV6_HEADER_ECHO structure 313
ICMPV6_HEADER_ERROR structure 314
ICMPV6_INFO_EREQ_CODE macro 311
ICMPV6_INFO_ERPL_CODE macro 311
ICMPV6_PACKET_TYPES enumeration 314
in_addr structure 70
in6_addr structure 82
INADDR_ANY macro 67
Include Files Processing 230
Introduction 3, 8, 28, 32, 52, 88, 102, 118, 138, 161, 173, 182, 189, 228, 294, 305, 318, 327, 342, 382, 432, 500, 504, 512, 515, 529, 550, 556, 599, 610, 651, 663, 669, 703
    Announce Module 28
    ARP Module 32
    Berkeley Module 52
    DHCP Module 88, 118
    DHCP Server Module 102
    DNS Module 138
    DNSS Module 161
    Dynamic DNS Module 173
    FTP Module 182
    HTTP Module 189
    HTTP Net Module 228
    ICMP Module 294
    ICMPv6 Module 305
    Iperf Module 318
    IPv4 Module 327
    IPv6 Module 342
    MAC Driver Module 382
    Manager Module 432
    NBNS Module 500
    NDP Module 504
    Reboot Module 512
    SMTP Module 515
    SMTPC Module 529
    SNMP Module 550
    SNTP Module 599
    TCP Module 610
    TCPIP Stack Library Overview 3
    TCPIP Stack Library Porting Guide 8
    Telnet Module 663
    TFTP Module 651
    UDP Module 669
    Zeroconf Module 703
INVALID_SOCKET macro 643
INVALID_TCP_PORT macro 67
INVALID_UDP_SOCKET macro 695
IP_ADD_MEMBERSHIP macro 74
IP_ADDR type 484
IP_ADDR_ANY macro 67
IP_ADDRESS_TYPE enumeration 485
ip_config.h 340
IP_DROP_MEMBERSHIP macro 74
IP_MULTI_ADDRESS union 485
IP_MULTICAST_IF macro 74
IP_MULTICAST_LOOP macro 74
IP_MULTICAST_TTL macro 74
IP_OPTIONS macro 75
IP_TOS macro 75
IP_TTL macro 75
IP_VERSION_4 macro 364
IP_VERSION_6 macro 364
Iperf Module 318
iperf_config.h 325
IPPROTO_ICMPV6 macro 75
IPPROTO_IP macro 68

IPPROTO_IPV6 macro 75
IPPROTO_TCP macro 68
IPPROTO_UDP macro 68
IPv4 Module 327
ipv4.h 340
IPV4_ADDR union 485
IPV4_FRAGMENT_INFO union 339
IPV4_HEADER structure 337
IPV4_HEADER_TYPE enumeration 337
IPV4_PACKET structure 338
IPv6 Module 342
ipv6.h 378
IPV6_ACTION enumeration 368
IPV6_ADDR union 486
IPV6_ADDR_HANDLE type 486
IPV6_ADDR_SCOPE enumeration 486
IPV6_ADDR_STRUCT structure 487
IPV6_ADDR_TYPE enumeration 487
IPV6_ADDRESS_POLICY structure 369
IPV6_ADDRESS_PREFERENCE enumeration 369
IPV6_ADDRESS_TYPE union 370
IPV6_CHECKSUM macro 76
ipv6_config.h 380
IPV6_DATA_DYNAMIC_BUFFER macro 364
IPV6_DATA_NETWORK_FIFO macro 364
IPV6_DATA_NONE macro 365
IPV6_DATA_PIC_RAM macro 365
IPV6_DATA_SEGMENT_HEADER structure 370
IPV6_EVENT_HANDLER type 371
IPV6_EVENT_TYPE enumeration 371
IPV6_FRAGMENT_HEADER structure 371
IPV6_HANDLE type 372
IPV6_HEADER structure 372
IPV6_HEADER_OFFSET_DEST_ADDR macro 365
IPV6_HEADER_OFFSET_NEXT_HEADER macro 365
IPV6_HEADER_OFFSET_PAYLOAD_LENGTH macro 365
IPV6_HEADER_OFFSET_SOURCE_ADDR macro 366
IPV6_JOIN_GROUP macro 76
IPV6_LEAVE_GROUP macro 76
IPV6_MULTICAST_HOPS macro 76
IPV6_MULTICAST_IF macro 76
IPV6_MULTICAST_LOOP macro 77
IPV6_NEXT_HEADER_TYPE enumeration 373
IPV6_NO_UPPER_LAYER_CHECKSUM macro 366
IPV6_PACKET structure 373
IPV6_PACKET_ACK_FNC type 375
IPV6_RX_FRAGMENT_BUFFER structure 375
IPV6_SEGMENT_TYPE enumeration 376
IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM macro 366
IPV6_TLV_HBHO_ROUTER_ALERT macro 366
IPV6_TLV_OPTION_TYPE union 376
IPV6_TLV_PAD_1 macro 366
IPV6_TLV_PAD_N macro 367
IPV6_TLV_UNREC_OPT_DISCARD_PP macro 367
IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC macro 367
IPV6_TLV_UNREC_OPT_DISCARD_SILENT macro 367
IPV6_TLV_UNREC_OPT_SKIP_OPTION macro 367
IPV6_ULA_FLAGS enumeration 376
IPV6_ULA_RESULT enumeration 377
IPV6_UNICAST_HOPS macro 77
IPV6_V6ONLY macro 77

L

Library Interface 36, 54, 90, 106, 126, 144, 165, 175, 185, 199, 243, 296, 307, 332, 348, 390, 436, 502, 509, 519, 535, 560, 603, 618, 655, 666, 675, 705
- ARP Module 36
- Berkeley Module 54
- DHCP Module 90
- DHCP Server Module 106
- DHCPv6 Module 126
- DNS Module 144
- DNSS Module 165
- Dynamic DNS Module 175
- FTP Module 185
- HTTP Module 199
- HTTP Net Module 243
- ICMP Module 296
- ICMPv6 Module 307
- IPv4 Module 332
- IPv6 Module 348
- MAC Driver Module 390
- Manager Module 436
- NBNS Module 502
- NDP Module 509
- SMTP Module 519
- SMTPC Module 535
- SNMP Module 560
- SNTP Module 603
- TCP Module 618
- Telnet Module 666
- TFTP Module 655
- UDP Module 675
- Zeroconf Module 705

Library Overview 33, 53, 88, 102, 118, 139, 162, 173, 182, 190, 229, 294, 306, 328, 343, 383, 433, 500, 505, 515, 529, 550, 599, 611, 651, 664, 670, 703
- ARP Module 33
- Berkeley Module 53
- DHCP Module 88
- DHCP Server Module 102
- DHCPv6 Module 118
- DNS Module 139
- DNSS Module 162
- Dynamic DNS Module 173
- FTP Module 182
- HTTP Module 190
- HTTP Net Module 229
- ICMP Module 294
- ICMPv6 Module 306
- IPv4 Module 328
- IPv6 Module 343
- MAC Driver Module 383
- Manager Module 433
- NBNS Module 500
- NDP Module 505
- SMTP Module 515

SMTPC Module 529
SNMP Module 550
SNTP Module 599
TCP Module 611
Telnet Module 664
TFTP Module 651
UDP Module 670
Zeroconf Module 703
linger structure 80
Link Local (Zeroconf) 704
listen function 59

M

MAC Driver Module 382
Main Program Changes 15
Manager Module 432
MAX_BSD_SOCKETS macro 53
mDNS 704
MDNSD_ERR_CODE enumeration 710
MIB Browsers 558
MIB Files 558
mpfs2.jar Utility 231
MPLAB Harmony TCP/IP Stack Access Changes 16
MPLAB Harmony TCP/IP Stack API Changes 9
MPLAB Harmony TCP/IP Stack Configuration 13
MPLAB Harmony TCP/IP Stack Configuration Changes 13
MPLAB Harmony TCP/IP Stack Design 9
MPLAB Harmony TCP/IP Stack Function Name Compliance 17
MPLAB Harmony TCP/IP Stack Heap Configuration 14
MPLAB Harmony TCP/IP Stack Initialization Changes 15
MPLAB Harmony TCP/IP Stack Key Features 8
MPLAB Harmony TCP/IP Stack SSL/RSA Usage 16
MPLAB Harmony TCP/IP Stack Storage Changes 13
MPLAB Harmony TCP/IP Stack Structure 8
MPLAB Harmony TCP/IP Stack Utilities 16

N

NBNS Module 500
nbns.h 503
nbns_config.h 503
NDP Module 504
ndp.h 510
ndp_config.h 510
Network Metrics 3
New MPLAB Harmony TCP/IP Stack API Functions 14
NO_DATA macro 81
NO_RECOVERY macro 81

P

Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack 16
Porting the MPLAB Harmony Drivers and Peripheral Library 23

R

Reboot Module 512
recv function 60
recvfrom function 60

S

send function 61
sendto function 61
Service Discovery 704
setsockopt function 63
SMTP Client Examples 515
SMTP Client Long Message Example 516
SMTP Client Short Message Example 516
SMTP Module 515
smtp.h 527
smtp_config.h 527
SMTP_CONNECT_ERROR macro 525
SMTP_RESOLVE_ERROR macro 525
SMTP_SUCCESS macro 525
SMTPC Module 529
smtpc.h 547
smtpc_config.h 548
SNMP Module 550
SNMP Server (Agent) 556
SNMP Traps 559
snmp.h 594
SNMP_COMMUNITY_TYPE enumeration 586
snmp_config.h 597
SNMP_END_OF_VAR macro 584
SNMP_GENERIC_TRAP_NOTIFICATION_TYPE enumeration 588
SNMP_ID type 586
SNMP_INDEX type 586
SNMP_INDEX_INVALID macro 585
SNMP_NON_MIB_RECD_INFO structure 587
SNMP_START_OF_VAR macro 585
SNMP_TRAP_IP_ADDRESS_TYPE enumeration 587
SNMP_V1 macro 585
SNMP_V2C macro 585
SNMP_V3 macro 585
SNMP_VAL union 587
SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE enumeration 589
snmpv3.h 596
snmpv3_config.h 598
SNMPV3_HMAC_HASH_TYPE enumeration 589
SNMPV3_PRIV_PROT_TYPE enumeration 589
SNTP Module 599
sntp.h 608
sntp_config.h 609
SO_BROADCAST macro 77
SO_DEBUG macro 77
SO_DONTROUTE macro 78
SO_KEEPALIVE macro 78
SO_LINGER macro 78
SO_OOBINLINE macro 78
SO_RCVBUF macro 78
SO_RCVLOWAT macro 79
SO_RCVTIMEO macro 79
SO_REUSEADDR macro 79
SO_SNDBUF macro 79
SO_SNDLOWAT macro 79
SO_SNDTIMEO macro 80
SOCK_DGRAM macro 68
SOCK_STREAM macro 68
sockaddr structure 70

SOCKADDR type 69
sockaddr_in structure 71
SOCKADDR_IN type 69
sockaddr_in6 structure 82
SOCKADDR_IN6 type 83
sockaddr_storage structure 82
socket function 62
SOCKET type 70
SOCKET_CNXN_IN_PROGRESS macro 69
SOCKET_DISCONNECTED macro 69
SOCKET_ERROR macro 69
SOL_SOCKET macro 80
SSI Processing 230
STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL enumeration 590
STD_BASED_SNMP_SECURITY_MODEL enumeration 591
STD_BASED_SNMPV3_SECURITY_LEVEL enumeration 591

T

TCP Changes 10
TCP Module 610
tcp.h 647
TCP/IP Stack Libraries Help 2
TCP/IP Stack Library Overview 3
TCP/IP Stack Library Porting Guide 8
TCP_ADJUST_FLAGS enumeration 640
tcp_config.h 648
TCP_NODELAY macro 80
TCP_OPTION_KEEP_ALIVE_DATA structure 644
TCP_OPTION_LINGER_DATA structure 640
TCP_OPTION_THRES_FLUSH_TYPE enumeration 643
TCP_PORT type 641
TCP_SOCKET type 641
TCP_SOCKET_INFO structure 641
TCP_SOCKET_OPTION enumeration 642
tcpip.h 498
tcpip_announce_config.h 31
TCPIP_ANNOUNCE_MAX_PAYLOAD macro 30
TCPIP_ANNOUNCE_TASK_RATE macro 29
TCPIP_ARP_CACHE_ENTRIES macro 34
TCPIP_ARP_CACHE_ENTRY_RETRIES macro 34
TCPIP_ARP_CACHE_PENDING_ENTRY_TMO macro 34
TCPIP_ARP_CACHE_PENDING_RETRY_TMO macro 34
TCPIP_ARP_CACHE_PERMANENT_QUOTA macro 35
TCPIP_ARP_CACHE_PURGE_QUANTA macro 35
TCPIP_ARP_CACHE_PURGE_THRESHOLD macro 35
TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO macro 35
TCPIP_ARP_CacheEntriesNoGet function 43
TCPIP_ARP_CacheThresholdSet function 44
TCPIP_ARP_ENTRY_QUERY structure 45
TCPIP_ARP_ENTRY_TYPE enumeration 46
TCPIP_ARP_EntryGet function 37
TCPIP_ARP_EntryQuery function 44
TCPIP_ARP_EntryRemove function 38
TCPIP_ARP_EntryRemoveAll function 45
TCPIP_ARP_EntryRemoveNet function 38
TCPIP_ARP_EntrySet function 42
TCPIP_ARP_EVENT_HANDLER type 46
TCPIP_ARP_EVENT_TYPE enumeration 47
TCPIP_ARP_GRATUITOUS_PROBE_COUNT macro 35
TCPIP_ARP_HANDLE type 47
TCPIP_ARP_HandlerDeRegister function 39
TCPIP_ARP_HandlerRegister function 40
TCPIP_ARP_IsResolved function 40
TCPIP_ARP_MODULE_CONFIG structure 49
TCPIP_ARP_OPERATION_TYPE enumeration 47
TCPIP_ARP_Probe function 41
TCPIP_ARP_Resolve function 41
TCPIP_ARP_RESULT enumeration 48
TCPIP_ARP_Task function 42
TCPIP_ARP_TASK_PROCESS_RATE macro 36
TCPIP_BSD_Socket function 65
TCPIP_DDNS_CHECKIP_SERVER macro 174
TCPIP_DDNS_LastIPGet function 175
TCPIP_DDNS_LastStatusGet function 176
TCPIP_DDNS_ServiceSet function 176
TCPIP_DDNS_Task function 177
TCPIP_DDNS_TASK_TICK_RATE macro 174
TCPIP_DDNS_UpdateForce function 176
TCPIP_DHCP_BOOT_FILE_NAME_SIZE macro 89
TCPIP_DHCP_CLIENT_ENABLED macro 114
TCPIP_DHCP_Disable function 91
TCPIP_DHCP_Enable function 91
TCPIP_DHCP_EVENT_HANDLER type 96
TCPIP_DHCP_EVENT_TYPE enumeration 97
TCPIP_DHCP_HANDLE type 97
TCPIP_DHCP_HandlerDeRegister function 92
TCPIP_DHCP_HandlerRegister function 92
TCPIP_DHCP_HOST_NAME_CALLBACK type 100
TCPIP_DHCP_HOST_NAME_SIZE macro 100
TCPIP_DHCP_HostNameCallbackRegister function 93
TCPIP_DHCP_INFO structure 98
TCPIP_DHCP_InfoGet function 112
TCPIP_DHCP_IsActive function 96
TCPIP_DHCP_IsBound function 95
TCPIP_DHCP_IsEnabled function 95
TCPIP_DHCP_IsServerDetected function 95
TCPIP_DHCP_MODULE_CONFIG structure 98
TCPIP_DHCP_Renew function 94
TCPIP_DHCP_Request function 94
TCPIP_DHCP_RequestTimeoutSet function 109
TCPIP_DHCP_STATUS enumeration 99
TCPIP_DHCP_STORE_BOOT_FILE_NAME macro 89
TCPIP_DHCP_Task function 109
TCPIP_DHCP_TASK_TICK_RATE macro 114
TCPIP_DHCP_TIMEOUT macro 114
TCPIP_DHCPS_ADDRESS_CONFIG structure 113
TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START macro 103
TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS macro 104
TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS macro 104
TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS macro 104
TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS macro 104
TCPIP_DHCPS_Disable function 107
TCPIP_DHCPS_Enable function 107

TCPIP_DHCPS_GetPoolEntries function 111
TCPIP_DHCPS_IsEnabled function 110
TCPIP_DHCPS_LEASE_DURATION macro 104
TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT macro 105
TCPIP_DHCPS_LEASE_ENTRY structure 112
TCPIP_DHCPS_LEASE_HANDLE type 113
TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK macro 105
TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO macro 105
TCPIP_DHCPS_LeaseEntryGet function 111
TCPIP_DHCPS_LeaseEntryRemove function 108
TCPIP_DHCPS_MODULE_CONFIG structure 115
TCPIP_DHCPS_POOL_ENTRY_TYPE enumeration 113
TCPIP_DHCPS_RemovePoolEntries function 108
TCPIP_DHCPS_Task function 110
TCPIP_DHCPS_TASK_PROCESS_RATE macro 105
TCPIP_DHCPV6_CLIENT_DUID_TYPE macro 120
TCPIP_DHCPV6_CLIENT_INFO structure 128
TCPIP_DHCPV6_CLIENT_PORT macro 120
TCPIP_DHCPV6_CLIENT_STATE enumeration 129
TCPIP_DHCPV6_ClientInfoGet function 128
TCPIP_DHCPV6_CONFIG_FLAGS enumeration 129
TCPIP_DHCPV6_DNS_SERVERS_NO macro 120
TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE macro 120
TCPIP_DHCPV6_DUID_TYPE enumeration 130
TCPIP_DHCPV6_EVENT_HANDLER type 130
TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE macro 121
TCPIP_DHCPV6_HANDLE type 131
TCPIP_DHCPV6_HandlerDeRegister function 127
TCPIP_DHCPV6_HandlerRegister function 127
TCPIP_DHCPV6_IA_EVENT union 131
TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO macro 121
TCPIP_DHCPV6_IA_INFO structure 131
TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME macro 121
TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME macro 121
TCPIP_DHCPV6_IA_STATE enumeration 132
TCPIP_DHCPV6_IA_SUBSTATE enumeration 133
TCPIP_DHCPV6_IA_TYPE enumeration 133
TCPIP_DHCPV6_IaInfoGet function 128
TCPIP_DHCPV6_IANA_DEFAULT_T1 macro 121
TCPIP_DHCPV6_IANA_DEFAULT_T2 macro 122
TCPIP_DHCPV6_IANA_DESCRIPTORS_NO macro 122
TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO macro 122
TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS macro 122
TCPIP_DHCPV6_IANA_SOLICIT_T1 macro 122
TCPIP_DHCPV6_IANA_SOLICIT_T2 macro 123
TCPIP_DHCPV6_IATA_DEFAULT_T1 macro 123
TCPIP_DHCPV6_IATA_DEFAULT_T2 macro 123
TCPIP_DHCPV6_IATA_DESCRIPTORS_NO macro 123
TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO macro 123
TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS macro 124
TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE macro 124
TCPIP_DHCPV6_MESSAGE_BUFFERS macro 124
TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE macro 124
TCPIP_DHCPV6_MODULE_CONFIG structure 133
TCPIP_DHCPV6_SERVER_PORT macro 124
TCPIP_DHCPV6_SERVER_STATUS_CODE enumeration 134
TCPIP_DHCPV6_SKIP_DAD_PROCESS macro 125
TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN macro 125
TCPIP_DHCPV6_Task function 127
TCPIP_DHCPV6_TASK_TICK_RATE macro 125
TCPIP_DHCPV6_USER_NOTIFICATION macro 125
TCPIP_DNS_CLIENT_ADDRESS_TYPE macro 142
TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL macro 142
TCPIP_DNS_CLIENT_CACHE_ENTRIES macro 141
TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO macro 141
TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS macro 141
TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS macro 141
TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO macro 142
TCPIP_DNS_CLIENT_INFO structure 156
TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO macro 143
TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN macro 143
TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES macro 143
TCPIP_DNS_CLIENT_MODULE_CONFIG structure 156
TCPIP_DNS_CLIENT_SERVER_TMO macro 141
TCPIP_DNS_CLIENT_TASK_PROCESS_RATE macro 142
TCPIP_DNS_CLIENT_USER_NOTIFICATION macro 142
TCPIP_DNS_ClientInfoGet function 149
TCPIP_DNS_ClientTask function 149
TCPIP_DNS_Disable function 145
TCPIP_DNS_Enable function 145
TCPIP_DNS_ENABLE_FLAGS enumeration 158
TCPIP_DNS_ENTRY_QUERY structure 157
TCPIP_DNS_EntryQuery function 150
TCPIP_DNS_EVENT_HANDLER type 154
TCPIP_DNS_EVENT_TYPE enumeration 154
TCPIP_DNS_GetIPAddressesNumber function 150
TCPIP_DNS_GetIPv4Addresses function 151
TCPIP_DNS_GetIPv6Addresses function 151
TCPIP_DNS_HANDLE type 155
TCPIP_DNS_HandlerDeRegister function 146
TCPIP_DNS_HandlerRegister function 146
TCPIP_DNS_IsEnabled function 147
TCPIP_DNS_IsNameResolved function 147
TCPIP_DNS_IsResolved function 153
TCPIP_DNS_RemoveAll function 148
TCPIP_DNS_RemoveEntry function 148
TCPIP_DNS_Resolve function 152
TCPIP_DNS_RESOLVE_TYPE enumeration 158
TCPIP_DNS_RESULT enumeration 155
TCPIP_DNS_Send_Query function 152
TCPIP_DNSS_AddressCntGet function 166
TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES macro 163
TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS macro 163
TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS macro 164
TCPIP_DNSS_CacheEntryRemove function 166
TCPIP_DNSS_Disable function 167
TCPIP_DNSS_Enable function 167
TCPIP_DNSS_EntryAdd function 168
TCPIP_DNSS_EntryGet function 169
TCPIP_DNSS_HOST_NAME_LEN macro 164
TCPIP_DNSS_IsEnabled function 168
TCPIP_DNSS_MODULE_CONFIG structure 170
TCPIP_DNSS_REPLY_BOARD_ADDR macro 164
TCPIP_DNSS_RESULT enumeration 170
TCPIP_DNSS_Task function 169
TCPIP_DNSS_TASK_PROCESS_RATE macro 164

TCPIP_DNSS_TTL_TIME macro 164
TCPIP_EMAC_ETH_OPEN_FLAGS macro 387
TCPIP_EMAC_PHY_ADDRESS macro 387
TCPIP_EMAC_PHY_CONFIG_FLAGS macro 388
TCPIP_EMAC_PHY_LINK_INIT_DELAY macro 388
TCPIP_EMAC_RX_BUFF_SIZE macro 388
TCPIP_EMAC_RX_DEDICATED_BUFFERS macro 389
TCPIP_EMAC_RX_DESCRIPTORS macro 388
TCPIP_EMAC_RX_FILTERS macro 390
TCPIP_EMAC_RX_FRAGMENTS macro 388
TCPIP_EMAC_RX_INIT_BUFFERS macro 389
TCPIP_EMAC_RX_LOW_FILL macro 390
TCPIP_EMAC_RX_LOW_THRESHOLD macro 390
TCPIP_EMAC_RX_MAX_FRAME macro 389
TCPIP_EMAC_TX_DESCRIPTORS macro 389
TCPIP_EVENT enumeration 482
TCPIP_EVENT_HANDLE type 483
TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE macro 183
TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE macro 183
TCPIP_FTP_MAX_CONNECTIONS macro 183
TCPIP_FTP_MODULE_CONFIG structure 186
TCPIP_FTP_PASSWD_LEN macro 184
TCPIP_FTP_PUT_ENABLED macro 184
TCPIP_FTP_ServerTask function 186
TCPIP_FTP_TIMEOUT macro 184
TCPIP_FTP_USER_NAME_LEN macro 184
TCPIP_FTPS_TASK_TICK_RATE macro 184
TCPIP_Helper_Base64Decode function 448
TCPIP_Helper_Base64Encode function 448
TCPIP_Helper_FormatNetBIOSName function 439
TCPIP_Helper_htonl function 439
TCPIP_Helper_htonll function 445
TCPIP_Helper_htons function 440
TCPIP_Helper_IPAddressToString function 440
TCPIP_Helper_IPv6AddressToString function 441
TCPIP_Helper_IsBcastAddress function 441
TCPIP_Helper_IsMcastAddress function 442
TCPIP_Helper_IsPrivateAddress function 442
TCPIP_Helper_MACAddressToString function 443
TCPIP_Helper_ntohl function 443
TCPIP_Helper_ntohl macro 483
TCPIP_Helper_ntohll function 445
TCPIP_Helper_ntohll macro 495
TCPIP_Helper_ntohs function 443
TCPIP_Helper_ntohs macro 484
TCPIP_Helper_SecurePortGetByIndex function 445
TCPIP_Helper_SecurePortSet function 446
TCPIP_Helper_StringToIPAddress function 444
TCPIP_Helper_StringToIPv6Address function 444
TCPIP_Helper_StringToMACAddress function 445
TCPIP_Helper_TCPSecurePortGet function 447
TCPIP_Helper_UDPSecurePortGet function 447
tcpip_helpers.h 497
TCPIP_HTTP_ActiveConnectionCountGet function 220
TCPIP_HTTP_ArgGet function 200
TCPIP_HTTP_CACHE_LEN macro 195
TCPIP_HTTP_CONFIG_FLAGS macro 195
TCPIP_HTTP_CurrentConnectionByteCountDec function 201
TCPIP_HTTP_CurrentConnectionByteCountGet function 202
TCPIP_HTTP_CurrentConnectionByteCountSet function 202
TCPIP_HTTP_CurrentConnectionCallbackPosGet function 203
TCPIP_HTTP_CurrentConnectionCallbackPosSet function 204
TCPIP_HTTP_CurrentConnectionDataBufferGet function 211
TCPIP_HTTP_CurrentConnectionFileGet function 211
TCPIP_HTTP_CurrentConnectionHandleGet function 220
TCPIP_HTTP_CurrentConnectionHasArgsGet function 204
TCPIP_HTTP_CurrentConnectionHasArgsSet function 205
TCPIP_HTTP_CurrentConnectionIndexGet function 221
TCPIP_HTTP_CurrentConnectionIsAuthorizedGet function 212
TCPIP_HTTP_CurrentConnectionIsAuthorizedSet function 212
TCPIP_HTTP_CurrentConnectionPostSmGet function 213
TCPIP_HTTP_CurrentConnectionPostSmSet function 205
TCPIP_HTTP_CurrentConnectionSocketGet function 214
TCPIP_HTTP_CurrentConnectionStatusGet function 206
TCPIP_HTTP_CurrentConnectionStatusSet function 214
TCPIP_HTTP_CurrentConnectionUserDataGet function 215
TCPIP_HTTP_CurrentConnectionUserDataSet function 216
TCPIP_HTTP_DEFAULT_FILE macro 195
TCPIP_HTTP_DEFAULT_LEN macro 196
TCPIP_HTTP_DYN_ARG_DCPT structure 278
TCPIP_HTTP_DYN_ARG_TYPE enumeration 279
TCPIP_HTTP_DYN_PRINT_RES enumeration 279
TCPIP_HTTP_DYN_VAR_DCPT structure 280
TCPIP_HTTP_DYN_VAR_FLAGS enumeration 280
TCPIP_HTTP_FILE_UPLOAD_ENABLE macro 198
TCPIP_HTTP_FILE_UPLOAD_NAME macro 198
TCPIP_HTTP_FileAuthenticate function 207
TCPIP_HTTP_FileInclude function 216
TCPIP_HTTP_GetExecute function 207
TCPIP_HTTP_MAX_CONNECTIONS macro 196
TCPIP_HTTP_MAX_DATA_LEN macro 196
TCPIP_HTTP_MAX_HEADER_LEN macro 196
TCPIP_HTTP_MIN_CALLBACK_FREE macro 196
TCPIP_HTTP_MODULE_CONFIG structure 224
TCPIP_HTTP_NET_ActiveConnectionCountGet function 244
TCPIP_HTTP_NET_ArgGet function 245
TCPIP_HTTP_NET_CACHE_LEN macro 234
TCPIP_HTTP_NET_CHUNK_RETRIES macro 239
TCPIP_HTTP_NET_CHUNKS_NUMBER macro 239
TCPIP_HTTP_NET_CONFIG_FLAGS macro 234
TCPIP_HTTP_NET_CONN_HANDLE type 281
TCPIP_HTTP_NET_ConnectionByteCountDec function 245
TCPIP_HTTP_NET_ConnectionByteCountGet function 247
TCPIP_HTTP_NET_ConnectionByteCountSet function 246
TCPIP_HTTP_NET_ConnectionCallbackPosGet function 248
TCPIP_HTTP_NET_ConnectionCallbackPosSet function 247
TCPIP_HTTP_NET_ConnectionDataBufferGet function 248
TCPIP_HTTP_NET_ConnectionDataBufferSizeGet function 273
TCPIP_HTTP_NET_ConnectionDiscard function 249
TCPIP_HTTP_NET_ConnectionDynamicDescriptors function 273
TCPIP_HTTP_NET_ConnectionFileAuthenticate function 250
TCPIP_HTTP_NET_ConnectionFileGet function 250
TCPIP_HTTP_NET_ConnectionFileInclude function 251
TCPIP_HTTP_NET_ConnectionFlush function 252
TCPIP_HTTP_NET_ConnectionGetExecute function 253
TCPIP_HTTP_NET_ConnectionHandleGet function 277

TCPIP_HTTP_NET_ConnectionHasArgsGet function 252
TCPIP_HTTP_NET_ConnectionHasArgsSet function 255
TCPIP_HTTP_NET_ConnectionIndexGet function 278
TCPIP_HTTP_NET_ConnectionIsAuthorizedGet function 253
TCPIP_HTTP_NET_ConnectionIsAuthorizedSet function 256
TCPIP_HTTP_NET_ConnectionNetHandle function 257
TCPIP_HTTP_NET_ConnectionPeek function 259
TCPIP_HTTP_NET_ConnectionPostExecute function 260
TCPIP_HTTP_NET_ConnectionPostNameRead function 254
TCPIP_HTTP_NET_ConnectionPostReadPair macro 287
TCPIP_HTTP_NET_ConnectionPostSmGet function 255
TCPIP_HTTP_NET_ConnectionPostSmSet function 261
TCPIP_HTTP_NET_ConnectionPostValueRead function 257
TCPIP_HTTP_NET_ConnectionRead function 262
TCPIP_HTTP_NET_ConnectionReadBufferSize function 262
TCPIP_HTTP_NET_ConnectionReadIsReady function 263
TCPIP_HTTP_NET_ConnectionSocketGet function 258
TCPIP_HTTP_NET_ConnectionStatusGet function 259
TCPIP_HTTP_NET_ConnectionStatusSet function 263
TCPIP_HTTP_NET_ConnectionStringFind function 264
TCPIP_HTTP_NET_ConnectionUserAuthenticate function 265
TCPIP_HTTP_NET_ConnectionUserDataGet function 260
TCPIP_HTTP_NET_ConnectionUserDataSet function 266
TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE macro 234
TCPIP_HTTP_NET_DEFAULT_FILE macro 234
TCPIP_HTTP_NET_DynAcknowledge function 266
TCPIP_HTTP_NET_DynamicWrite function 267
TCPIP_HTTP_NET_DynamicWriteString function 268
TCPIP_HTTP_NET_DynPrint function 268
TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER macro 235
TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER macro 235
TCPIP_HTTP_NET_DYNVAR_MAX_LEN macro 235
TCPIP_HTTP_NET_DYNVAR_PROCESS macro 239
TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES macro 239
TCPIP_HTTP_NET_EVENT_TYPE enumeration 281
TCPIP_HTTP_NET_EventReport function 269
TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES macro 239
TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE macro 240
TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER macro 240
TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE macro 235
TCPIP_HTTP_NET_FILE_UPLOAD_NAME macro 235
TCPIP_HTTP_NET_FILENAME_MAX_LEN macro 240
TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE macro 236
TCPIP_HTTP_NET_IO_RESULT enumeration 283
TCPIP_HTTP_NET_MAX_CONNECTIONS macro 236
TCPIP_HTTP_NET_MAX_DATA_LEN macro 236
TCPIP_HTTP_NET_MAX_HEADER_LEN macro 236
TCPIP_HTTP_NET_MAX_RECURSE_LEVEL macro 237
TCPIP_HTTP_NET_MODULE_CONFIG structure 283
TCPIP_HTTP_NET_MODULE_FLAGS enumeration 284
TCPIP_HTTP_NET_READ_STATUS enumeration 285
TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE macro 237
TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE macro 237
TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE macro 237
TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER macro 240
TCPIP_HTTP_NET_SSI_CMD_MAX_LEN macro 240
TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE macro 241
TCPIP_HTTP_NET_SSI_PROCESS macro 241
TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER macro 241
TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH macro 241
TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH macro 242
TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER macro 242
TCPIP_HTTP_NET_SSINotification function 274
TCPIP_HTTP_NET_SSIVariableDelete function 274
TCPIP_HTTP_NET_SSIVariableGet function 275
TCPIP_HTTP_NET_SSIVariableGetByIndex function 275
TCPIP_HTTP_NET_SSIVariableSet function 276
TCPIP_HTTP_NET_SSIVariablesNumberGet function 277
TCPIP_HTTP_NET_STATUS enumeration 285
TCPIP_HTTP_NET_Task function 271
TCPIP_HTTP_NET_TASK_RATE macro 238
TCPIP_HTTP_NET_TIMEOUT macro 238
TCPIP_HTTP_NET_URLDecode function 271
TCPIP_HTTP_NET_USE_AUTHENTICATION macro 238
TCPIP_HTTP_NET_USE_COOKIES macro 238
TCPIP_HTTP_NET_USE_POST macro 238
TCPIP_HTTP_NET_USER_CALLBACK structure 286
TCPIP_HTTP_NET_USER_HANDLE type 287
TCPIP_HTTP_NET_UserHandlerDeregister function 272
TCPIP_HTTP_NET_UserHandlerRegister function 272
TCPIP_HTTP_NO_AUTH_WITHOUT_SSL macro 199
TCPIP_HTTP_PostExecute function 208
TCPIP_HTTP_PostNameRead function 217
TCPIP_HTTP_PostReadPair macro 219
TCPIP_HTTP_PostValueRead function 217
TCPIP_HTTP_Print_varname function 209
TCPIP_HTTP_SKT_RX_BUFF_SIZE macro 197
TCPIP_HTTP_SKT_TX_BUFF_SIZE macro 197
TCPIP_HTTP_SSI_ATTR_DCPT structure 288
TCPIP_HTTP_SSI_NOTIFY_DCPT structure 289
TCPIP_HTTP_Task function 219
TCPIP_HTTP_TASK_RATE macro 198
TCPIP_HTTP_TIMEOUT macro 197
TCPIP_HTTP_URLDecode function 218
TCPIP_HTTP_USE_AUTHENTICATION macro 197
TCPIP_HTTP_USE_COOKIES macro 197
TCPIP_HTTP_USE_POST macro 198
TCPIP_HTTP_UserAuthenticate function 210
TCPIP_HTTPS_DEFAULT_FILE macro 198
TCPIP_ICMP_CallbackDeregister function 297
TCPIP_ICMP_CallbackRegister function 297
TCPIP_ICMP_CLIENT_USER_NOTIFICATION macro 295
TCPIP_ICMP_ECHO_REQUEST structure 302
TCPIP_ICMP_ECHO_REQUEST_RESULT enumeration 303
TCPIP_ICMP_ECHO_REQUEST_TIMEOUT macro 295
TCPIP_ICMP_EchoRequest function 300
TCPIP_ICMP_EchoRequestCancel function 301
TCPIP_ICMP_EchoRequestSend function 298
TCPIP_ICMP_MODULE_CONFIG structure 302
TCPIP_ICMP_REQUEST_HANDLE type 303
TCPIP_ICMP_Task function 299
TCPIP_ICMP_TASK_TICK_RATE macro 295
TCPIP_ICMPV6_CallbackDeregister function 308
TCPIP_ICMPV6_CallbackRegister function 310
TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION macro 315

TCPIP_ICMPV6_EchoRequestSend function 309
TCPIP_ICMPV6_Flush function 308
TCPIP_ICMPV6_HeaderEchoRequestPut function 309
TCPIP_ICMPV6_PutHeaderEchoReply macro 311
TCPIP_IPERF_MAX_INSTANCES macro 324
TCPIP_IPERF_RX_BUFFER_SIZE macro 323
TCPIP_IPERF_TIMING_ERROR_MARGIN macro 324
TCPIP_IPERF_TX_BUFFER_SIZE macro 324
TCPIP_IPERF_TX_QUEUE_LIMIT macro 324
TCPIP_IPERF_TX_WAIT_TMO macro 324
TCPIP_IPV4_FILTER_TYPE enumeration 338
TCPIP_IPV4_FRAGMENT_MAX_NUMBER macro 330
TCPIP_IPV4_FRAGMENT_MAX_STREAMS macro 330
TCPIP_IPV4_FRAGMENT_TIMEOUT macro 331
TCPIP_IPV4_FRAGMENTATION macro 331
TCPIP_IPV4_MacPacketTransmit function 336
TCPIP_IPV4_MODULE_CONFIG structure 339
TCPIP_IPV4_PacketFilterClear function 335
TCPIP_IPV4_PacketFilterSet function 335
TCPIP_IPV4_PacketFormatTx function 336
TCPIP_IPV4_PacketGetDestAddress function 332
TCPIP_IPV4_PacketGetSourceAddress function 333
TCPIP_IPV4_PacketTransmit function 333
TCPIP_IPV4_SelectSourceInterface function 334
TCPIP_IPV4_Task function 334
TCPIP_IPV4_TASK_TICK_RATE macro 331
TCPIP_IPV6_AddressUnicastRemove function 349
TCPIP_IPV6_ArrayGet function 350
TCPIP_IPV6_ArrayPutHelper function 350
TCPIP_IPV6_DASSourceAddressSelect function 351
TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE macro 344
TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME macro 344
TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT macro 344
TCPIP_IPV6_DEFAULT_LINK_MTU macro 345
TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME macro 345
TCPIP_IPV6_DefaultRouterDelete function 362
TCPIP_IPV6_DefaultRouterGet function 363
TCPIP_IPV6_DestAddressGet function 352
TCPIP_IPV6_DestAddressSet function 352
TCPIP_IPV6_Flush function 353
TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT macro 345
TCPIP_IPV6_Get function 353
TCPIP_IPV6_HandlerDeregister function 354
TCPIP_IPV6_HandlerRegister function 354
TCPIP_IPV6_INIT_TASK_PROCESS_RATE macro 345
TCPIP_IPV6_InterfaceIsReady function 355
TCPIP_IPV6_MINIMUM_LINK_MTU macro 345
TCPIP_IPV6_MODULE_CONFIG structure 377
TCPIP_IPV6_MTU_INCREASE_TIMEOUT macro 506
TCPIP_IPV6_MulticastListenerAdd function 355
TCPIP_IPV6_MulticastListenerRemove function 356
TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME macro 506
TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME macro 506
TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT macro 507
TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT macro 507
TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY macro 507
TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS macro 507
TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT macro 507
TCPIP_IPV6_NDP_REACHABLE_TIME macro 508
TCPIP_IPV6_NDP_RETRANS_TIMER macro 508
TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL macro 508
TCPIP_IPV6_NDP_TASK_TIMER_RATE macro 508
TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS macro 508
TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT macro 346
TCPIP_IPV6_PacketFree function 356
TCPIP_IPV6_PayloadSet function 357
TCPIP_IPV6_Put function 357
TCPIP_IPV6_PutArray macro 368
TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT macro 346
TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT macro 346
TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT macro 346
TCPIP_IPV6_RouterAddressAdd function 362
TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE macro 346
TCPIP_IPV6_SourceAddressGet function 358
TCPIP_IPV6_SourceAddressSet function 358
TCPIP_IPV6_Task function 363
TCPIP_IPV6_TASK_PROCESS_RATE macro 347
TCPIP_IPV6_TxIsPutReady function 359
TCPIP_IPV6_TxPacketAllocate function 360
TCPIP_IPV6_ULA_NTP_ACCESS_TMO macro 347
TCPIP_IPV6_ULA_NTP_VALID_WINDOW macro 347
TCPIP_IPV6_UnicastAddressAdd function 361
TCPIP_IPV6_UniqueLocalUnicastAddressAdd function 360
TCPIP_LOCAL_MASK_TYPE enumeration 481
tcpip_mac.h 429
TCPIP_MAC_ACTION enumeration 404
TCPIP_MAC_ADDR structure 405
TCPIP_MAC_Close function 393
tcpip_mac_config.h 430
TCPIP_MAC_ConfigGet function 399
TCPIP_MAC_DATA_SEGMENT structure 405
TCPIP_MAC_Deinitialize function 394
TCPIP_MAC_ETHERNET_HEADER structure 407
TCPIP_MAC_EVENT enumeration 407
TCPIP_MAC_EventAcknowledge function 400
TCPIP_MAC_EventF type 408
TCPIP_MAC_EventMaskSet function 400
TCPIP_MAC_EventPendingGet function 401
TCPIP_MAC_HANDLE type 409
TCPIP_MAC_HEAP_CallocF type 409
TCPIP_MAC_HEAP_CallocFDbg type 424
TCPIP_MAC_HEAP_FreeF type 410
TCPIP_MAC_HEAP_FreeFDbg type 424
TCPIP_MAC_HEAP_HANDLE type 410
TCPIP_MAC_HEAP_MallocF type 410
TCPIP_MAC_HEAP_MallocFDbg type 424
TCPIP_MAC_INIT structure 422
TCPIP_MAC_Initialize function 393
TCPIP_MAC_LINK_MTU enumeration 428
TCPIP_MAC_LinkCheck function 396
TCPIP_MAC_MODULE_CTRL structure 411
TCPIP_MAC_Open function 392
TCPIP_MAC_PACKET type 411
TCPIP_MAC_PACKET_ACK_FUNC type 413
TCPIP_MAC_PACKET_FLAGS enumeration 414

TCPIP_MAC_PACKET_RX_STAT structure 414
TCPIP_MAC_PacketRx function 403
TCPIP_MAC_PacketTx function 402
TCPIP_MAC_PARAMETERS structure 422
TCPIP_MAC_ParametersGet function 395
TCPIP_MAC_PKT_ACK_RES enumeration 415
TCPIP_MAC_PKT_AckF type 416
TCPIP_MAC_PKT_AckFDbg type 423
TCPIP_MAC_PKT_AllocF type 417
TCPIP_MAC_PKT_AllocFDbg type 423
TCPIP_MAC_PKT_FreeF type 417
TCPIP_MAC_PKT_FreeFDbg type 423
TCPIP_MAC_POWER_MODE enumeration 418
TCPIP_MAC_Process function 398
TCPIP_MAC_PROCESS_FLAGS enumeration 418
TCPIP_MAC_RegisterStatisticsGet function 398
TCPIP_MAC_Reinitialize function 394
TCPIP_MAC_RES enumeration 418
TCPIP_MAC_RX_FILTER_TYPE enumeration 427
TCPIP_MAC_RX_STATISTICS structure 421
TCPIP_MAC_RxFilterHashTableEntrySet function 402
TCPIP_MAC_SEGMENT_FLAGS enumeration 419
TCPIP_MAC_STATISTICS_REG_ENTRY structure 424
TCPIP_MAC_StatisticsGet function 395
TCPIP_MAC_Status function 396
TCPIP_MAC_SYNCH_REQUEST enumeration 425
TCPIP_MAC_SynchReqF type 426
TCPIP_MAC_Tasks function 397
TCPIP_MAC_TX_STATISTICS structure 421
TCPIP_MAC_TYPE enumeration 425
tcpip_manager.h 496
TCPIP_MDNS_ServiceDeregister function 706
TCPIP_MDNS_ServiceRegister function 706
TCPIP_MDNS_ServiceUpdate function 707
TCPIP_MDNS_Task function 708
TCPIP_MODULE_MAC_ID enumeration 426
TCPIP_MODULE_MAC_MRF24W_CONFIG structure 420
TCPIP_MODULE_MAC_MRF24WN_CONFIG structure 427
TCPIP_MODULE_MAC_PIC32INT_CONFIG structure 420
TCPIP_MODULE_SIGNAL enumeration 491
TCPIP_MODULE_SIGNAL_FUNC type 492
TCPIP_MODULE_SIGNAL_HANDLE type 492
TCPIP_MODULE_SignalFunctionDeregister function 456
TCPIP_MODULE_SignalFunctionRegister function 456
TCPIP_MODULE_SignalGet function 457
TCPIP_NBNS_MODULE_CONFIG structure 502
TCPIP_NBNS_Task function 502
TCPIP_NBNS_TASK_TICK_RATE macro 501
TCPIP_NDP_NborReachConfirm function 509
TCPIP_NET_HANDLE type 481
TCPIP_NETWORK_CONFIG structure 488
TCPIP_NETWORK_CONFIG_FLAGS enumeration 489
TCPIP_NTP_DEFAULT_CONNECTION_TYPE macro 600
TCPIP_NTP_DEFAULT_IF macro 600
TCPIP_NTP_EPOCH macro 601
TCPIP_NTP_FAST_QUERY_INTERVAL macro 601
TCPIP_NTP_MAX_STRATUM macro 601
TCPIP_NTP_QUERY_INTERVAL macro 601
TCPIP_NTP_REPLY_TIMEOUT macro 601
TCPIP_NTP_RX_QUEUE_LIMIT macro 602
TCPIP_NTP_SERVER macro 602
TCPIP_NTP_SERVER_MAX_LENGTH macro 602
TCPIP_NTP_TASK_TICK_RATE macro 602
TCPIP_NTP_TIME_STAMP_TMO macro 602
TCPIP_NTP_VERSION macro 603
tcpip_reboot_config.h 514
TCPIP_REBOOT_MESSAGE macro 513
TCPIP_REBOOT_SAME_SUBNET_ONLY macro 513
TCPIP_REBOOT_TASK_TICK_RATE macro 513
TCPIP_SMTP_ArrayPut function 520
TCPIP_SMTP_CLIENT_MESSAGE structure 526
TCPIP_SMTP_CLIENT_MODULE_CONFIG structure 525
TCPIP_SMTP_ClientTask function 524
TCPIP_SMTP_Flush function 521
TCPIP_SMTP_IsBusy function 521
TCPIP_SMTP_IsPutReady function 521
TCPIP_SMTP_MailSend function 524
TCPIP_SMTP_MAX_WRITE_SIZE macro 519
TCPIP_SMTP_Put function 522
TCPIP_SMTP_PutIsDone function 522
TCPIP_SMTP_SERVER_REPLY_TIMEOUT macro 518
TCPIP_SMTP_StringPut function 523
TCPIP_SMTP_TASK_TICK_RATE macro 518
TCPIP_SMTP_UsageBegin function 523
TCPIP_SMTP_UsageEnd function 524
TCPIP_SMTP_WRITE_READY_SPACE macro 518
TCPIP_SMTPC_ATTACH_BUFFER structure 537
TCPIP_SMTPC_ATTACH_FILE structure 537
TCPIP_SMTPC_ATTACH_TYPE enumeration 538
TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE macro 531
TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE macro 531
TCPIP_SMTPC_CLIENT_MESSAGE_DATE macro 531
TCPIP_SMTPC_ENCODE_TYPE enumeration 538
TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT macro 531
TCPIP_SMTPC_MAIL_CONNECTIONS macro 531
TCPIP_SMTPC_MAIL_FLAGS enumeration 539
TCPIP_SMTPC_MAIL_MESSAGE structure 540
TCPIP_SMTPC_MAIL_RETRIES macro 532
TCPIP_SMTPC_MailMessage function 535
TCPIP_SMTPC_MESSAGE_CALLBACK type 541
TCPIP_SMTPC_MESSAGE_HANDLE type 541
TCPIP_SMTPC_MESSAGE_QUERY structure 542
TCPIP_SMTPC_MESSAGE_REPORT structure 542
TCPIP_SMTPC_MESSAGE_RESULT enumeration 543
TCPIP_SMTPC_MESSAGE_STATUS enumeration 544
TCPIP_SMTPC_MESSAGE_WARNING enumeration 545
TCPIP_SMTPC_MessageQuery function 536
TCPIP_SMTPC_MODULE_CONFIG structure 546
TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE macro 532
TCPIP_SMTPC_SERVER_DATA_TIMEOUT macro 532
TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE macro 532
TCPIP_SMTPC_SERVER_REPLY_CALLBACK type 546
TCPIP_SMTPC_SERVER_REPLY_TIMEOUT macro 533
TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT macro 533
TCPIP_SMTPC_SKT_RX_BUFF_SIZE macro 533
TCPIP_SMTPC_SKT_TX_BUFF_SIZE macro 533

TCPIP_SMTPC_Task function 536
TCPIP_SMTPC_TASK_TICK_RATE macro 533
TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT macro 534
TCPIP_SMTPC_USE_MAIL_COMMAND macro 534
TCPIP_SNMP_AuthTrapFlagGet function 575
TCPIP_SNMP_AuthTrapFlagSet function 575
TCPIP_SNMP_BIB_FILE_NAME macro 552
TCPIP_SNMP_ClientGetNet function 564
TCPIP_SNMP_COMMUNITY_CONFIG structure 592
TCPIP_SNMP_COMMUNITY_MAX_LEN macro 552
TCPIP_SNMP_ExactIndexGet function 564
TCPIP_SNMP_IsTrapEnabled function 576
TCPIP_SNMP_IsValidCommunity function 565
TCPIP_SNMP_IsValidLength function 565
TCPIP_SNMP_MAX_COMMUNITY_SUPPORT macro 552
TCPIP_SNMP_MAX_MSG_SIZE macro 553
TCPIP_SNMP_MAX_NON_REC_ID_OID macro 553
TCPIP_SNMP_MibIDSet function 566
TCPIP_SNMP_MODULE_CONFIG structure 593
TCPIP_SNMP_NextIndexGet function 566
TCPIP_SNMP_NOTIFY_COMMUNITY_LEN macro 553
TCPIP_SNMP_NotifyIsReady function 562
TCPIP_SNMP_NotifyPrepare function 562
TCPIP_SNMP_OID_MAX_LEN macro 553
TCPIP_SNMP_ReadCommunityGet function 567
TCPIP_SNMP_ReadCommunitySet function 579
TCPIP_SNMP_RecordIDValidation function 567
TCPIP_SNMP_SendFailureTrap function 568
TCPIP_SNMP_SocketIDGet function 580
TCPIP_SNMP_SocketIDSet function 580
TCPIP_SNMP_Task function 581
TCPIP_SNMP_TASK_PROCESS_RATE macro 553
TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN macro 554
TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE macro 554
TCPIP_SNMP_TRAP_TABLE_SIZE macro 554
TCPIP_SNMP_TrapInterFaceSet function 569
TCPIP_SNMP_TRAPMibIDGet function 569
TCPIP_SNMP_TrapSendFlagGet function 570
TCPIP_SNMP_TrapSendFlagSet function 570
TCPIP_SNMP_TrapSpecificNotificationGet function 571
TCPIP_SNMP_TrapSpecificNotificationSet function 571
TCPIP_SNMP_TrapTimeGet function 563
TCPIP_SNMP_TRAPTypeGet function 576
TCPIP_SNMP_TRAPv1Notify function 577
TCPIP_SNMP_TRAPv2Notify function 577
TCPIP_SNMP_ValidateTrapIntf function 578
TCPIP_SNMP_VarbindGet function 572
TCPIP_SNMP_VarbindSet function 573
TCPIP_SNMP_WriteCommunityGet function 574
TCPIP_SNMP_WriteCommunitySet function 579
TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN macro 554
TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
macro 554
TCPIP_SNMPV3_EngineUserDataBaseGet function 581
TCPIP_SNMPV3_EngineUserDataBaseSet function 582
TCPIP_SNMPv3_Notify function 584
TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN macro 555
TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
macro 555
TCPIP_SNMPV3_TARGET_ENTRY_CONFIG structure 593
TCPIP_SNMPv3_TrapConfigDataGet function 583
TCPIP_SNMPV3_TrapTypeGet function 578
TCPIP_SNMPV3_USER_SECURITY_NAME_LEN macro 555
TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE macro 555
TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration 592
TCPIP_SNMPV3_USM_MAX_USER macro 555
TCPIP_SNMPV3_USM_USER_CONFIG structure 594
TCPIP_SNTP_ConnectionInitiate function 604
TCPIP_SNTP_ConnectionParamSet function 606
TCPIP_SNTP_LastErrorGet function 604
TCPIP_SNTP_MODULE_CONFIG structure 607
TCPIP_SNTP_RESULT enumeration 607
TCPIP_SNTP_Task function 606
TCPIP_SNTP_TimeStampGet function 605
TCPIP_SNTP_UTCSecondsGet function 605
TCPIP_STACK_Deinitialize function 453
TCPIP_STACK_EVENT_HANDLER type 483
TCPIP_STACK_EventsPendingGet function 463
TCPIP_STACK_HandlerDeregister function 451
TCPIP_STACK_HandlerRegister function 451
TCPIP_STACK_IF_NAME_97J60 macro 493
TCPIP_STACK_IF_NAME_ALIAS_ETH macro 484
TCPIP_STACK_IF_NAME_ALIAS_UNK macro 484
TCPIP_STACK_IF_NAME_ALIAS_WLAN macro 484
TCPIP_STACK_IF_NAME_ENCJ60 macro 493
TCPIP_STACK_IF_NAME_ENCJ600 macro 493
TCPIP_STACK_IF_NAME_MRF24W macro 493
TCPIP_STACK_IF_NAME_MRF24WN macro 496
TCPIP_STACK_IF_NAME_NONE macro 493
TCPIP_STACK_IF_NAME_PIC32INT macro 494
TCPIP_STACK_IF_POWER_DOWN macro 494
TCPIP_STACK_IF_POWER_FULL macro 494
TCPIP_STACK_IF_POWER_LOW macro 494
TCPIP_STACK_IF_POWER_NONE macro 494
TCPIP_STACK_IndexToNet function 458
TCPIP_STACK_INIT structure 489
TCPIP_STACK_Initialize function 452
TCPIP_STACK_InitializeDataGet function 466
TCPIP_STACK_MACObjectGet function 455
TCPIP_STACK_MODULE enumeration 490
TCPIP_STACK_MODULE_CONFIG structure 491
TCPIP_STACK_ModuleConfigGet function 476
TCPIP_STACK_NetAddress function 468
TCPIP_STACK_NetAddressBcast function 469
TCPIP_STACK_NetAddressDnsPrimary function 470
TCPIP_STACK_NetAddressDnsPrimarySet function 470
TCPIP_STACK_NetAddressDnsSecond function 471
TCPIP_STACK_NetAddressDnsSecondSet function 471
TCPIP_STACK_NetAddressGateway function 472
TCPIP_STACK_NetAddressGatewaySet function 473
TCPIP_STACK_NetAddressMac function 473
TCPIP_STACK_NetAddressMacSet function 474
TCPIP_STACK_NetAddressSet function 475

TCPIP_STACK_NetAliasNameGet function 465
TCPIP_STACK_NetBIOSName function 458
TCPIP_STACK_NetBiosNameSet function 459
TCPIP_STACK_NetConfigGet function 479
TCPIP_STACK_NetConfigSet function 480
TCPIP_STACK_NetDefaultGet function 459
TCPIP_STACK_NetDefaultSet function 460
TCPIP_STACK_NetDown function 468
TCPIP_STACK_NetHandleGet function 460
TCPIP_STACK_NetIndexGet function 461
TCPIP_STACK_NetIPv6AddressGet function 475
TCPIP_STACK_NetIsLinked function 466
TCPIP_STACK_NetIsReady function 464
TCPIP_STACK_NetIsUp function 467
TCPIP_STACK_NetMACIdGet function 477
TCPIP_STACK_NetMACRegisterStatisticsGet function 464
TCPIP_STACK_NetMACStatisticsGet function 478
TCPIP_STACK_NetMask function 461
TCPIP_STACK_NetNameGet function 462
TCPIP_STACK_NetUp function 467
TCPIP_STACK_NumberOfNetworksGet function 463
TCPIP_STACK_SetLocalMasks function 449
TCPIP_STACK_SetLocalMasksType function 450
TCPIP_STACK_Status function 478
TCPIP_STACK_Task function 453
TCPIP_STACK_USE_BASE64_DECODE macro 195
TCPIP_STACK_USE_DHCPV6_CLIENT macro 125
TCPIP_STACK_VERSION_MAJOR macro 495
TCPIP_STACK_VERSION_MINOR macro 495
TCPIP_STACK_VERSION_PATCH macro 495
TCPIP_STACK_VERSION_STR macro 495
TCPIP_STACK_VersionGet function 454
TCPIP_STACK_VersionStrGet function 454
TCPIP_TCP_Abort function 624
TCPIP_TCP_ArrayFind function 636
TCPIP_TCP_ArrayGet function 635
TCPIP_TCP_ArrayPeek function 635
TCPIP_TCP_ArrayPut function 632
TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL macro 613
TCPIP_TCP_Bind function 621
TCPIP_TCP_ClientOpen function 620
TCPIP_TCP_Close function 620
TCPIP_TCP_CLOSE_WAIT_TIMEOUT macro 613
TCPIP_TCP_Connect function 621
TCPIP_TCP_DELAYED_ACK_TIMEOUT macro 613
TCPIP_TCP_Discard function 638
TCPIP_TCP_Disconnect function 624
TCPIP_TCP_DYNAMIC_OPTIONS macro 617
TCPIP_TCP_FifoRxFreeGet function 638
TCPIP_TCP_FifoRxFullGet macro 639
TCPIP_TCP_FifoSizeAdjust function 639
TCPIP_TCP_FifoTxFreeGet function 634
TCPIP_TCP_FifoTxFullGet function 633
TCPIP_TCP_FIN_WAIT_2_TIMEOUT macro 614
TCPIP_TCP_Find function 636
TCPIP_TCP_Flush function 633
TCPIP_TCP_Get function 637
TCPIP_TCP_GetIsReady function 634
TCPIP_TCP_IsConnected function 623
TCPIP_TCP_IsReady function 629
TCPIP_TCP_KEEP_ALIVE_TIMEOUT macro 614
TCPIP_TCP_MAX_RETRIES macro 614
TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL macro 614
TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL macro 614
TCPIP_TCP_MAX_SEG_SIZE_TX macro 615
TCPIP_TCP_MAX_SOCKETS macro 615
TCPIP_TCP_MAX_SYN_RETRIES macro 615
TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES macro 615
TCPIP_TCP_MODULE_CONFIG structure 643
TCPIP_TCP_MSL_TIMEOUT macro 617
TCPIP_TCP_OptionsGet function 625
TCPIP_TCP_OptionsSet function 625
TCPIP_TCP_Peek function 637
TCPIP_TCP_Put function 631
TCPIP_TCP_PutIsReady function 631
TCPIP_TCP_QUIET_TIME macro 617
TCPIP_TCP_RemoteBind function 622
TCPIP_TCP_ServerOpen function 619
TCPIP_TCP_SIGNAL_FUNCTION type 644
TCPIP_TCP_SIGNAL_HANDLE type 645
TCPIP_TCP_SIGNAL_TYPE enumeration 645
TCPIP_TCP_SignalHandlerDeregister function 628
TCPIP_TCP_SignalHandlerRegister function 628
TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE macro 615
TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE macro 616
TCPIP_TCP_SocketInfoGet function 626
TCPIP_TCP_SocketNetGet function 627
TCPIP_TCP_SocketNetSet function 627
TCPIP_TCP_SocketsNumberGet function 630
TCPIP_TCP_SocketTraceSet function 630
TCPIP_TCP_START_TIMEOUT_VAL macro 616
TCPIP_TCP_STATE enumeration 646
TCPIP_TCP_StringPut function 631
TCPIP_TCP_Task function 629
TCPIP_TCP_TASK_TICK_RATE macro 616
TCPIP_TCP_WasReset function 623
TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL macro 616
TCPIP_TELNET_MAX_CONNECTIONS macro 665
TCPIP_TELNET_MODULE_CONFIG structure 667
TCPIP_TELNET_PASSWORD macro 665
TCPIP_TELNET_Task function 667
TCPIP_TELNET_TASK_TICK_RATE macro 665
TCPIP_TELNET_USERNAME macro 665
TCPIP_TFTP_CMD_TYPE enumeration 658
TCPIP_TFTPC_ARP_TIMEOUT macro 653
TCPIP_TFTPC_CMD_PROCESS_TIMEOUT macro 653
TCPIP_TFTPC_DEFAULT_IF macro 653
TCPIP_TFTPC_EVENT_HANDLER type 659
TCPIP_TFTPC_EVENT_TYPE enumeration 659
TCPIP_TFTPC_FILENAME_LEN macro 653
TCPIP_TFTPC_GetEventNotification function 656
TCPIP_TFTPC_HANDLE type 660
TCPIP_TFTPC_HandlerDeRegister function 656
TCPIP_TFTPC_HandlerRegister function 657
TCPIP_TFTPC_HOSTNAME_LEN macro 654
TCPIP_TFTPC_MAX_RETRIES macro 654

TCPIP_TFTPC_MODULE_CONFIG structure 659
TCPIP_TFTPC_OPERATION_RESULT enumeration 660
TCPIP_TFTPC_SetCommand function 658
TCPIP_TFTPC_SetServerAddress function 657
TCPIP_TFTPC_Task function 655
TCPIP_TFTPC_TASK_TICK_RATE macro 654
TCPIP_TFTPC_USER_NOTIFICATION macro 654
TCPIP_UDP_ArrayGet function 693
TCPIP_UDP_ArrayPut function 690
TCPIP_UDP_BcastIPV4AddressSet function 684
TCPIP_UDP_Bind function 678
TCPIP_UDP_ClientOpen function 677
TCPIP_UDP_Close function 680
TCPIP_UDP_DestinationIPAddressSet function 685
TCPIP_UDP_DestinationPortSet function 686
TCPIP_UDP_Discard function 694
TCPIP_UDP_Disconnect function 686
TCPIP_UDP_Flush function 692
TCPIP_UDP_Get function 693
TCPIP_UDP_GetIsReady function 692
TCPIP_UDP_IsConnected function 678
TCPIP_UDP_IsOpened macro 677
TCPIP_UDP_MAX_SOCKETS macro 672
TCPIP_UDP_MODULE_CONFIG structure 698
TCPIP_UDP_OptionsGet function 680
TCPIP_UDP_OptionsSet function 681
TCPIP_UDP_Put function 691
TCPIP_UDP_PutIsReady function 689
TCPIP_UDP_RemoteBind function 679
TCPIP_UDP_RxOffsetSet function 694
TCPIP_UDP_ServerOpen function 676
TCPIP_UDP_SIGNAL_FUNCTION type 698
TCPIP_UDP_SIGNAL_HANDLE type 698
TCPIP_UDP_SIGNAL_TYPE enumeration 699
TCPIP_UDP_SignalHandlerDeregister function 687
TCPIP_UDP_SignalHandlerRegister function 687
TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT macro 673
TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT macro 673
TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE macro 673
TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE macro 673
TCPIP_UDP_SOCKET_POOL_BUFFERS macro 673
TCPIP_UDP_SocketInfoGet function 682
TCPIP_UDP_SocketNetGet function 682
TCPIP_UDP_SocketNetSet function 683
TCPIP_UDP_SocketsNumberGet function 688
TCPIP_UDP_SourceIPAddressSet function 684
TCPIP_UDP_StringPut function 690
TCPIP_UDP_Task function 688
TCPIP_UDP_TxCountGet function 691
TCPIP_UDP_TxOffsetSet function 683
TCPIP_UDP_TxPutIsReady function 689
TCPIP_UDP_USE_POOL_BUFFERS macro 674
TCPIP_UDP_USE_RX_CHECKSUM macro 674
TCPIP_UDP_USE_TX_CHECKSUM macro 674
TCPIP_ZCLL_Disable function 708
TCPIP_ZCLL_Enable function 709
TCPIP_ZCLL_IsEnabled function 709
TCPIP_ZCLL_Task function 710
Telnet Module 663
telnet.h 667
telnet_config.h 668
TFTP Client Server Communication 651
TFTP Module 651
TFTP_CMD_GET_TYPE enumeration member 658
TFTP_CMD_NONE enumeration member 658
TFTP_CMD_PUT_TYPE enumeration member 658
tftpc.h 661
tftpc_config.h 662
TRY_AGAIN macro 81

U

UDP Changes 11
UDP Module 669
udp.h 699
udp_config.h 701
UDP_PORT type 695
UDP_SOCKET type 695
UDP_SOCKET_BCAST_TYPE enumeration 696
UDP_SOCKET_INFO structure 696
UDP_SOCKET_OPTION enumeration 697
Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony TCP/IP Stack 8
Using Iperf 318
Using the Library 28, 32, 52, 88, 102, 118, 138, 161, 173, 182, 189, 228, 294, 305, 318, 327, 342, 382, 432, 500, 504, 512, 515, 529, 550, 599, 610, 651, 663, 669, 703
- Announce Module 28
- ARP Module 32
- Berkeley Module 52
- DHCP Module 88
- DHCP Server Module 102
- DHCPv6 Module 118
- DNS Module 138
- DNSS Module 161
- Dynamic DNS Module 173
- FTP Module 182
- HTTP Module 189
- HTTP Net Module 228
- ICMP Module 294
- ICMPv6 Module 305
- IPv4 Module 327
- IPv6 Module 342
- MAC Driver Module 382
- Manager Module 432
- NBNS Module 500
- NDP Module 504
- Reboot Module 512
- SMTP Module 515
- SMTPC Module 529
- SNMP Module 550
- SNTP Module 599
- TCP Module 610
- Telnet Module 663
- TFTP Module 651
- UDP Module 669
- Zeroconf Module 703

Z

ZCLL_MODULE_CONFIG structure 710
zero_conf_link_local.h 711
zero_conf_multicast_dns.h 711
Zeroconf Enabled Environments 704
Zeroconf Module 703