TCPIP Sockets Interface User Guide

Version 3.40

For use with TCP/IP Sockets Interface module versions 4.01 and above

Date: 21-Feb-2018 16:33

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

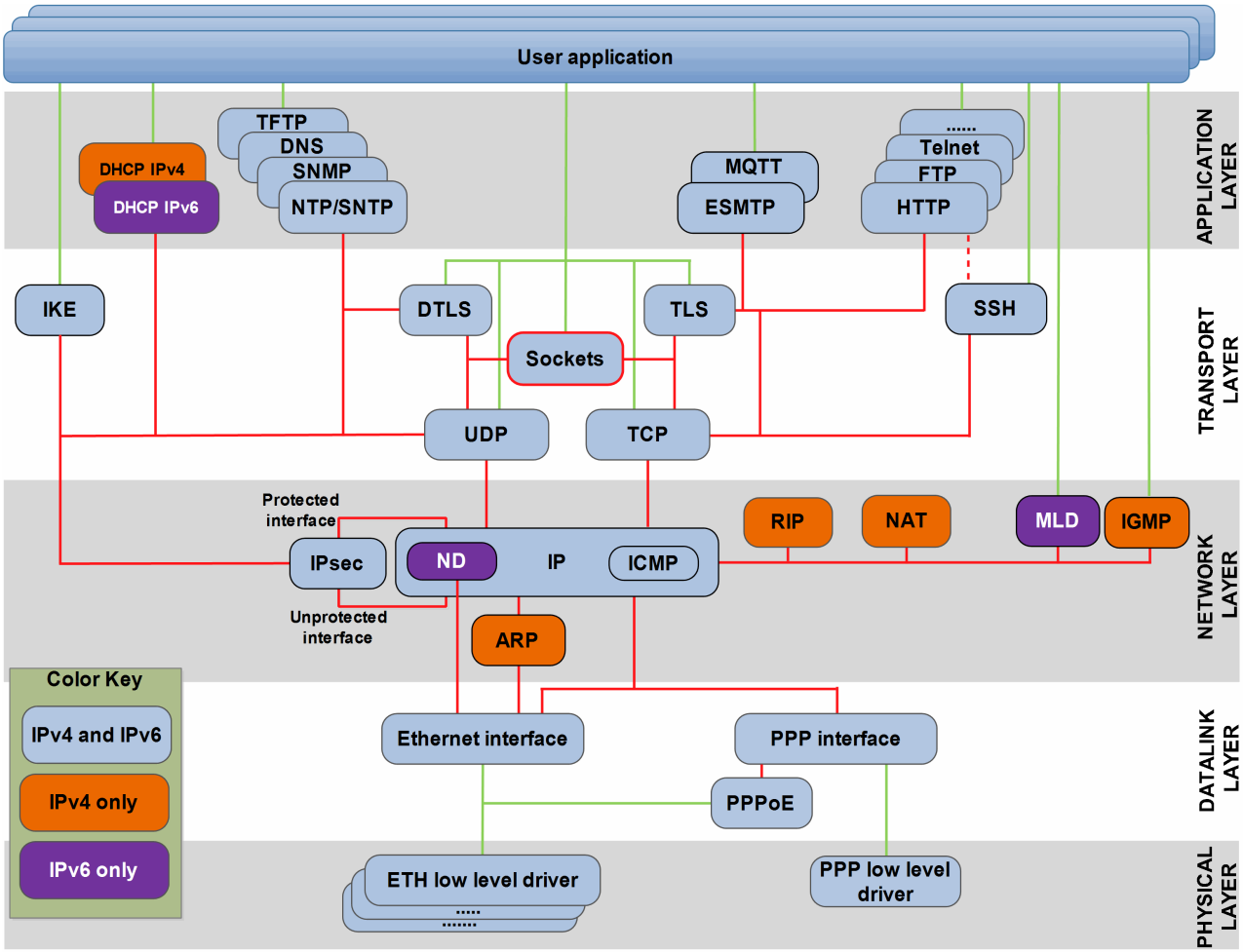
- Introduction – describes the main elements of the module. This section includes a diagram showing the position of this module within HCC's TCP/IP stack.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1 Introduction

This guide is for those who want to add the optional IP Sockets module to the HCC TCP/IP Suite. When integrated, the HCC IP Sockets module allows an application to communicate across a TCP/IP network using the industry standard BSD Sockets system calls. This manual describes how to use the HCC IP Sockets module.

User applications can use the native interfaces of the system simultaneously with the Sockets API as long as they operate on different UDP or TCP ports.

The Sockets module is part of the HCC MISRA-compliant TCP/IP stack, as shown below, and is designed specifically for use with it. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Complies with the HCC MISRA-compliant TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compliant with BSD Sockets.
- IPv6 operation complies with RFC 2553.
- Provides a standard interface for legacy applications to use.
- Allows portability of applications across Sockets-compliant systems.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
ip_socket	The Sockets package.
mip_base	The TCP/IP Dual Stack base package.

Documents

For an overview of the HCC TCP/IP stack software, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC TCP/IP Dual Stack System User Guide

This is the core document that describes the complete TCP/IP stack. It covers both IPv4 and IPv6 systems.

HCC TCP/IP Sockets Interface User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To download earlier manuals, see TCP/IP PDFs.
- For the history of changes made to the package code itself, see History: ip_socket.

The current version of this manual is 3.40. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
3.40	2018-02-21	4.01	Improved description of functions' return codes. Changed names of many macros, definitions and structures (added prefix *SOCKET_* or *socket_*).
3.30	2017-11-15	3.34	Added options SO_KEEPALIVE, SO_BROADCAST, SO_REUSEADDR, and SO_NODELAY.
3.20	2017-10-11	3.32	Added PSP macros to *PSP Porting*.
3.10	2017-09-18	3.30	Changed definition of IP_V4_MAP_IP_V6_ADDR_PREFIX_SIZE. Added functions to *PSP Porting*.
3.00	2017-09-05	3.29	Corrected *Packages* list.
2.90	2017-06-20	3.29	New *Change History* format.
2.80	2017-03-20	3.29	Updated network diagram.
2.70	2017-01-16	3.28	Updated network diagram.
2.60	2016-04-20	3.26	Added function group tables to API section.
2.50	2015-11-12	3.21	Added SO_LINGER option. Updated to work with IP base major version 6.
2.40	2015-08-18	3.19	Extended **socket_select()**.
2.30	2015-04-28	3.19	Added **socket_htonl()**, **socket_htons()**, **socket_ntohs()**, **socket_ntohl()**.
2.20	2015-03-31	3.19	Added software change history to manual.
2.10	2014-08-22	3.19	Reorganized *System Overview* section.
2.00	2014-05-14	3.13	First online version.

2 Source File List

This section lists and describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_ip_socket.h** is the only file that should be included by an application using this module.

This file defines all API function calls and related structures that are available to the user. For details of these API functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_ip_socket.h** contains all the configurable parameters of the UDP system. Configure these as required. For details of these options, see Configuration Options.

2.3 Sockets System

The file **src/ip/stack/socket/socket.c** contains the TCP/IP Sockets source code. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_ip_socket.h** contains the software version number of this release of the package. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_ip_socket.h**. This section lists the available options and their default values.

SOCK_TABLE_SIZE

The maximum number of sockets that the system provides to the user. The default is 8. Increasing the value increases the amount of RAM allocated by the system.

SOCK_DGRAM_PORT_BASE

The port base for DGRAM connections. This is used when **socket_sendto()** is called without Bind. The default is 10600.

The port range is from SOCK_DGRAM_PORT_BASE to SOCK_DGRAM_PORT_BASE + SOCK_TABLE_SIZE - 1.

SOCK_INET_NTOA

Set this to 1 to make the **socket_inet_ntoa()** function available. The default is 0.

> **Note:** If the above option is set, the RAM usage is increased by 16 * IP_MAX_TASK.

SOCK_GETHOSTBYNAME

Set this to 1 to make the **socket_gethostbyaddr()** and **socket_gethostbyname()** functions available. The default is 0.

> **Note:** If the above option is set, RAM usage is increased by about (36 + IP_MAX_FQDN_SIZE) * IP_MAX_TASK.

SOCK_TASK_ERRNO

If the socket layer is used with an RTOS, the default of 1 makes error numbers task-specific. Set this to 0 to disable this.

SOCK_RCV_INFO_ENABLE

Set this to 1 to allow Virtual LAN (VLAN) TCI and destination/source MAC address information requests for the last received packet on a socket. The default is 0.

> **Note:** This option is only valid for Ethernet interfaces.

4 Using the API

This section describes the HCC Sockets Application Programming Interface (API) then shows time sequence diagrams of the communication between the server and the client, using the TCP and UDP sockets.

The Sockets API may be used to access TCP ports and UDP ports. It can be used in parallel with the TCP /IP stack's native APIs, as long as the APIs do not simultaneously use the same ports.

The Sockets API has these advantages:

- It is an industry standard BSD Sockets interface. This means applications written for other platforms that use BSD sockets can be used unchanged.
- Applications can use their own buffers to read and write data to the IP stack, although inevitably this means that there is a copy in the interface.

The main drawbacks of using the Sockets API, compared to the native API, are:

- The native API yields slightly better performance and allows the possibility of zero copy transfers.
- The native API is designed with modern and strict coding standards in mind, and its components are consequently easier to interface to applications requiring a more rigorous implementation.

4.1 Sockets Usage Summary

The Sockets interface functions are used as follows:

- In all cases the user application must communicate by using a socket.
- The user application must first allocate a socket for the communication.
- Once allocated, the socket handle is used for all accesses including connection establishment, data transfer, and connection tear-down.

Note: HCC supplies reference application code. Using this is recommended to ensure the interface is used correctly.

Blocking behavior is as follows:

- If the system is running without an RTOS then all socket calls are non-blocking.
- If the system is running with an RTOS then calls on open sockets will block unless they have been explicitly switched to non-blocking using the socket_ioctl() function.

Broadcasts operate as follows:

- If the IP address field is set to 255.255.255.255, the packet is broadcast on the network interface configured as the default gateway.

- If a specific subnetwork is specified (for example, 192.168.0.255), the broadcast is sent on the interface with that network address.

4.2 HCC Sockets Function Name Mapping

The names used by standard Sockets functions can cause conflicts in systems. For example, a function name of "**close**" occurs in many systems. As a consequence HCC has renamed the standard functions as shown in the table below. All function parameters are used exactly as in the Sockets specification. If required, these function names can be easily remapped using a definition file.

HCC Name	Standard Sockets Name
socket_open	socket
socket_close	close
socket_bind	bind
socket_listen	listen
socket_accept	accept
socket_connect	connect
socket_recv	recv
socket_recvfrom	recvfrom
socket_send	send
socket_sendto	sendto
socket_getopt	getsockopt
socket_setopt	setsockopt
socket_select	select
socket_poll	poll

4.3 Data Paths

Creating Data Paths

TCP

There are two main ways of establishing TCP socket connections: as a server and as a client.

For a client the typical connection setup sequence is:

- **socket_open()** to allocate a socket.
- **socket_connect()** to connect to a specific remote server using the allocated socket.

For a server the typical connection setup sequence is:

- **socket_open()** to allocate a socket.
- **socket_bind()** to associate a TCP port with the allocated socket.
- **socket_listen()** to wait for a remote client to connect to this service.
- **socket_accept()** to wait for a remote port to connect to this server.

UDP

Both server and client use the same means of establishing UDP socket connections. The typical connection setup sequence is:

- **socket_open()** to allocate a socket.
- **socket_bind()** to associate a UDP port with the allocated socket.

Managing Data Paths

For TCP data paths, use **socket_recv()** and **socket_send()** to transfer data.

For UDP data paths, use **socket_recvfrom()** and **socket_sendto()** to transfer data.

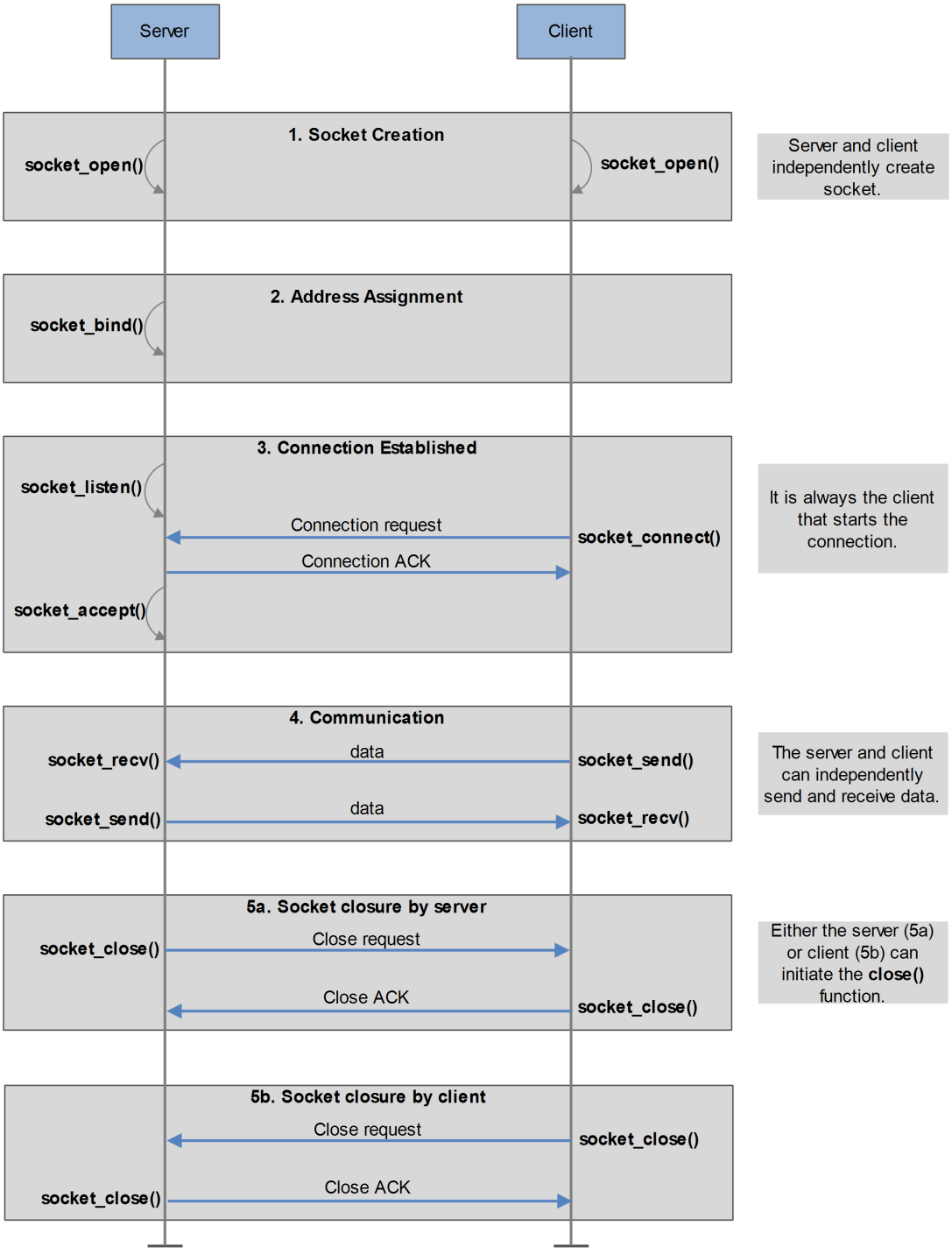
Sockets can be monitored for multiple events by using **socket_select()** and **socket_poll()**.

Closing Data Paths

Terminate TCP and UDP data paths by using **socket_close()**.

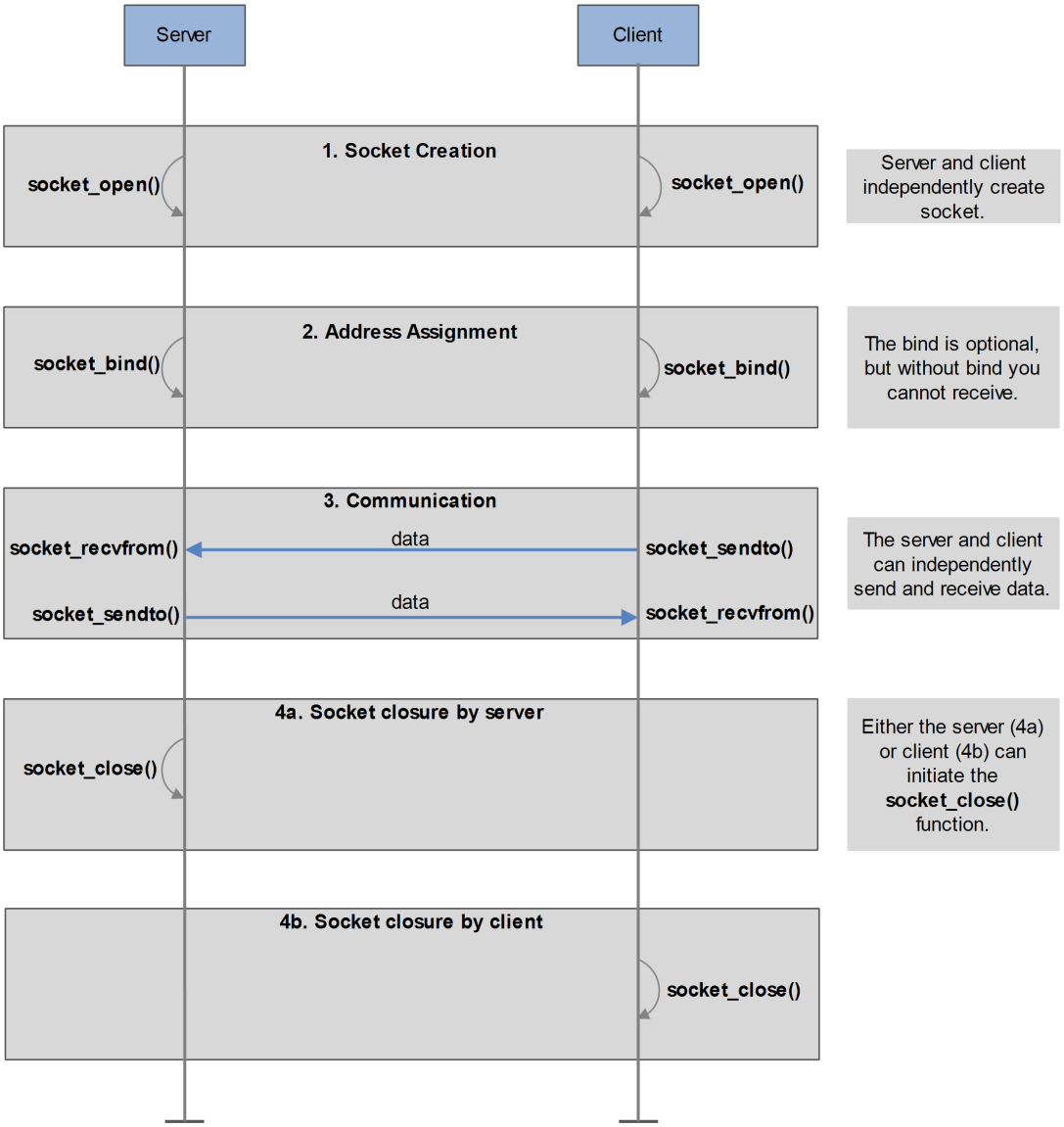
4.4 TCP Time Sequence

The following time sequence diagram shows the communication between the server and the client, using the TCP socket. The communication schema is connection-oriented: the two parties use the socket in stream mode.



4.5 UDP Time Sequence

The following time sequence diagram shows the communication between the server and the client, using the UDP socket. The communication schema is connectionless: the two parties use the socket in datagram mode.



5 Application Programming Interface

This section describes all the Application Programming Interface (API) functions. It includes all the functions that are available to an application program.

5.1 Module Management

The functions are the following:

Function	Description
socket_init()	Initializes the module and allocates the required resources.
socket_start()	Starts the module.
socket_stop()	Stops the module.
socket_delete()	Deletes the module and releases the resources it used.

You must ensure that:

- **socket_init()** is called before any other socket function.
- **socket_start()** is called before the socket API is used.
- No API function is used after **socket_stop()** is called.
- After **socket_delete()**, **socket_init()** is called before any other socket call.

Note:

- The module management calls use the TCP/IP stack error codes rather than socket-specific error codes. This allows easier management of error codes.
- All HCC module management functions are designed to be non-reentrant. The system designer must ensure that these calls are made systematically and cannot pre-empt each other. Typically you do this by performing all management of a particular module from a single task.

socket_init

Use this function to initialize the Sockets module, allocating the resources it requires.

> **Note:** This must be the first Sockets function called.

Format

```
t_ip_ret socket_init ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERR_OS	OS resource creation failure.

socket_start

Use this function to start the Sockets module.

This function must be called before any standard socket API call is made.

> **Note:** Call **socket_init()** before this function.

Format

```
t_ip_ret socket_start ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
Else	See Error Codes.

socket_stop

Use this function to stop the Sockets module.

After the Sockets module is stopped, no standard socket API calls may be called until another call to **socket_start()** is made.

Format

```
t_ip_ret socket_stop ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
Else	See Error Codes.

socket_delete

Use this function to delete the Sockets module.

This function may only be called when the Sockets module is in the stopped state. It frees any resources allocated to the module by **socket_init()**.

> **Note:** After **socket_delete()**, a new call to **socket_init()** is required before any other function is called.

Format

```
t_ip_ret socket_delete ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERR_OS	OS resource delete error.

5.2 Sockets Functions

Function	Description
socket_accept()	Starts listening on a TCP port.
socket_bind()	Opens a TCP/UDP port and associates it with the socket descriptor.
socket_close()	Closes the port (or, in the TCP case, connection) associated with the socket.
socket_connect()	Connects to a server using TCP.
socket_get_errno()	Gets the socket error number for the caller task.
socket_gethostbyaddr()	Gets a pointer to the *socket_hostent* structure, based on the IP address.
socket_gethostbyname()	Gets a pointer to the *socket_hostent* structure, based on the host name.
socket_getopt()	Gets the value of the requested socket option.
socket_htonl()	Converts a 32 bit integer from machine byte order to network byte order.
socket_htons()	Converts a 16 bit integer from machine byte order to network byte order.
socket_inet_aton()	Converts an ASCII string to an IPv4 address.
socket_inet_ntoa()	Converts an IP address to ASCII.
socket_ioctl()	Accesses the SOCKET_FIONBIO and SOCKET_FIONREAD commands.
socket_listen()	Starts listening on a TCP port.
socket_ntohl()	Converts a 32 bit integer from network byte order to machine (host) byte order.
socket_ntohs()	Converts a 16 bit integer from network byte order to machine (host) byte order.
socket_open()	Allocates and initializes a new socket descriptor.
socket_poll()	Waits until either timeout or a specified event occurs.
socket_recv()	Receives data over a TCP connection.
socket_recvfrom()	Receives a UDP datagram from the port associated with the socket.
socket_select()	Waits until a specified receive/transmission event or exception occurs.
socket_send()	Sends data over a TCP connection.
socket_sendto()	Sends a UDP datagram to a remote host.
socket_setopt()	Sets the value of the requested socket option.

socket_accept

Use this function to start listening on a TCP port.

The call order should be: socket, bind, listen, accept.

Format

```
int socket_accept (
    int                         sockfd,
    struct socket_sockaddr *    p_client_addr,
    socket_socklen_t *          p_addrlen )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_client_addr	Where to write the client address.	socket_sockaddr *
p_addrlen	Where to write the length of the client address.	socket_socklen_t *

Return values

Return value	Description
Number of socket descriptor.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_bind

Use this function to open a TCP/UDP port and associate it with the socket descriptor.

Format

```
int socket_bind (
    int                              sockfd,
    const struct socket_sockaddr *   p_my_addr,
    socket_socklen_t                 addrlen )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_my_addr	The local host address.	socket_sockaddr *
addrlen	The length of the address structure.	socket_socklen_t

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_close

Use this function to close the port (or, in the TCP case, connection) associated with the socket.

Format

```
int socket_close ( int sockfd )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_connect

Use this function to connect to a server using TCP.

Format

```
int socket_connect (
    int                              sockfd,
    const struct socket_sockaddr *   p_server_addr,
    socket_socklen_t                 addrlen )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_server_addr	The structure holding the server address.	socket_sockaddr *
addrlen	The length of the *p_server_addr* structure.	socket_socklen_t

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_get_errno

Use this function to get the socket error number for the caller task.

Format

```
int socket_get_errno ( int * p_errno )
```

Arguments

Argument	Description	Type
p_errno	Where to write the last error number.	int *

Return values

Return value	Description
SOCKET_SUCCESS	Last error is valid.
SOCKET_ERROR	No last error available.

socket_gethostbyaddr

Use this function to get a pointer to the *socket_hostent* structure, based on the IP address.

Format

```
struct socket_hostent * socket_gethostbyaddr (
    const void *    addr,
    int             len,
    int             type )
```

Arguments

Argument	Description	Type
addr	A pointer to the IP address (structure socket_in_addr).	void *
len	The length of the IP address (the size of the structure socket_in_addr).	int
type	Either SOCKET_AF_INET or SOCKET_AF_INET6.	int

Return values

Return value	Description
A pointer to the *socket_hostent* structure.	Successful execution.
NULL	See Error Codes.

socket_gethostbyname

Use this function to get the pointer to the *socket_hostent* structure, based on the host name.

Format

```
struct socket_hostent * socket_gethostbyname ( const char * name )
```

Arguments

Argument	Description	Type
name	The name of the host.	char *

Return values

Return value	Description
A pointer to the *socket_hostent* structure.	Successful execution.
NULL	See Error Codes.

socket_getopt

Use this function to get the value of the requested socket option.

Format

```
int socket_getopt(
    int                  sockfd,
    int                  level,
    int                  option_name,
    void *               p_option_value,
    socket_socklen_t *   p_option_len )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
level	The socket option level.	int
option_name	The socket option identifier.	int
p_option_value	A pointer to the buffer with the socket option value.	void *
p_option_len	A pointer to the socket option length.	socket_socklen_t *

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_htonl

Use this function to convert a 32 bit integer from machine (host) byte order to network byte order.

htonl stands for "host-to-network long".

Format

```
unsigned int socket_htonl( unsigned int host_addr )
```

Arguments

Argument	Description	Type
host_addr	An integer in host byte order.	unsigned int

Return value

The integer in network byte order.

Note: This function is supplied for BSD compatibility reasons. HCC converts this to a NULL macro because internally HCC handles all network address and port assignments natively.

socket_htons

Use this function to convert a 16 bit integer from machine (host) byte order to network byte order.

htons stands for "host-to-network short".

Format

```
unsigned short socket_htons( unsigned short host_port )
```

Arguments

Argument	Description	Type
host_port	An integer in host byte order.	unsigned short

Return value

The integer in network byte order.

Note: This function is supplied for BSD compatibility reasons. HCC converts this to a NULL macro because internally HCC handles all network address and port assignments natively.

socket_inet_aton

Use this function to convert an ASCII string to an IPv4 address.

> **Note:** This function does not support IPv6 addresses.

Format

```
int socket_inet_aton (
    const char *              p_ascii,
    struct socket_in_addr *   p_addr )
```

Arguments

Argument	Description	Type
p_ascii	A pointer to the string to convert.	char *
p_addr	Where to write the resulting IPv4 address.	socket_in_addr *

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_inet_ntoa

Use this function to convert an IP address to ASCII.

Format

```
char * socket_inet_ntoa ( struct socket_in_addr addr )
```

Arguments

Argument	Description	Type
addr	The IP address to convert.	socket_in_addr

Return values

Return value	Description
A pointer to the ASCII form of the IP address.	Successful execution.
NULL	Operation failed.

socket_ioctl

Use this function to access the SOCKET_FIONBIO and SOCKET_FIONREAD commands.

- Use SOCKET_FIONBIO to set the blocking mode of the socket. If the argument pointer content is zero, this sets the port to blocking. If it is non-zero, the port is non-blocking. Note that if an RTOS is not used, the socket is always non-blocking.
- Use SOCKET_FIONREAD to pass a pointer (*p_arg*) to a location to store the number of bytes available to be read from the socket.

Format

```
int socket_ioctl (
    int               sockfd,
    int               cmd,
    unsigned long *   p_arg )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
cmd	SOCKET_FIONBIO or SOCKET_FIONREAD.	int
p_arg	This depends on the *cmd*: • For SOCKET_FIONBIO use this to supply a pointer to the input argument. • For SOCKET_FIONREAD this returns a pointer to the output argument.	unsigned long *

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_listen

Use this function to start listening on a TCP port.

The call order should be: socket, bind, listen.

Format

```
int socket_listen (
    int   sockfd,
    int   backlog )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
backlog	The maximum number of connections on the port.	int

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

socket_ntohl

Use this function to convert a 32 bit integer from network byte order to machine (host) byte order.

ntohl stands for "network-to-host long".

Format

```
unsigned int socket_ntohl( unsigned int net_addr )
```

Arguments

Argument	Description	Type
net_addr	An integer in network byte order.	unsigned int

Return value

The integer in host byte order.

Note: This function is supplied for BSD compatibility reasons. HCC converts this to a NULL macro because internally HCC handles all network address and port assignments natively.

socket_ntohs

Use this function to convert a 16 bit integer from network byte order to machine (host) byte order.

ntohs stands for "network-to-host short".

Format

```
unsigned short socket_ntohs( unsigned short net_port )
```

Arguments

Argument	Description	Type
net_port	An integer in network byte order.	unsigned short

Return value

The integer in host byte order.

Note: This function is supplied for BSD compatibility reasons. HCC converts this to a NULL macro because internally HCC handles all network address and port assignments natively.

socket_open

Use this function to allocate and initialize a new socket descriptor.

Format

```
int socket_open (
    int   domain,
    int   type,
    int   protocol )
```

Arguments

Argument	Description	Type
domain	The socket domain; this should always be SOCKET_AF_INET or SOCKET_AF_INET6.	int
type	The socket type (SOCKET_SOCK_DGRAM or SOCKET_SOCK_STREAM).	int
protocol	Not used.	int

Return values

Return value	Description
Socket descriptor number.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_poll

Use this function to wait until either timeout, or any of the events specified by the *p_fds*, occurs.

Format

```
int socket_poll (
    t_pollfd *     p_fds,
    unsigned int   nfds,
    int            timeout )
```

Arguments

Argument	Description	Type
p_fds	A pointer to the array of pollfd structures.	t_pollfd *
nfds	The number of entries in the *p_fds* array.	unsigned int
timeout	The time in milliseconds to wait for a result. Use -1 to wait forever.	int

Return values

Return value	Description
The number of sockets where an event occurred.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_recv

Use this function to receive data over a TCP connection.

Format

```
int socket_recv (
    int      sockfd,
    void *   p_buffer,
    int      length,
    int      flags)
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_buffer	A pointer to the data buffer.	void *
length	The size of the buffer in bytes.	int
flags	Not used.	int

Return values

Return value	Description
The number of bytes received.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_recvfrom

Use this function to receive a UDP datagram from the port associated with the socket.

Format

```
int socket_recvfrom (
    int                         sockfd,
    void *                      p_buffer,
    int                         length,
    int                         flags,
    struct socket_sockaddr *    remote_address,
    socket_socklen_t *          p_addrlen )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_buffer	A pointer to the data buffer.	void *
length	The size of the buffer in bytes.	int
flags	Not used.	int
remote_address	Where to write the address of the remote node.	socket_sockaddr *
p_addrlen	Where to write the length of the remote address.	socket_socklen_t *

Return values

Return value	Description
The number of bytes received.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_select

Use this function to wait until one of the following occurs:

- Data is received on any socket in the *p_readfds* set.
- Data is transmitted on any socket in the *p_writefds* set.
- An exception event occurs on any socket in the *p_exceptfds* set.

Format

```
int socket_select (
    int                                 nfds,
    t_fd_set *                          p_readfds,
    t_fd_set *                          p_writefds,
    t_fd_set *                          p_exceptfds,
    const struct socket_timeval *       p_timeout )
```

Arguments

Argument	Description	Type
nfds	The number of entries in a set.	int
p_readfds	A pointer to the set of socket descriptors to be monitored for incoming data.	t_fd_set *
p_writefds	A pointer to the set of socket descriptors to be monitored for completion of a send operation.	t_fd_set *
p_exceptfds	A pointer to the set of socket descriptors to be monitored for exception events. Currently socket disconnection is the only exception event supported.	t_fd_set *
p_timeout	A pointer to a *socket_timeval* structure. If either of the following applies, this call waits forever: • The pointer to the *socket_timeval* structure is NULL. • Both elements of the structure (seconds and microseconds) are set to -1.	socket_timeval *

Return values

Return value	Description
The number of socket descriptors ready for input /output. If this is 0 it means the timeout expired.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_send

Use this function to send data over a TCP connection.

Format

```
int socket_send (
    int             sockfd,
    const void *    p_buffer,
    int             length,
    int             flags )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_buffer	A pointer to the data buffer.	void *
length	The size of the buffer in bytes.	int
flags	Not used.	int

Return Values

Return value	Description
The number of bytes sent.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_sendto

Use this function to send a UDP datagram to a remote host.

This function allocates a maximum of one protocol buffer, copies data to it, and sends it.

Format

```
int socket_sendto (
    int                                 sockfd,
    const void *                        p_buffer,
    int                                 length,
    int                                 flags,
    const struct socket_sockaddr *      p_dest_addr,
    socket_socklen_t                    dest_len )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
p_buffer	A pointer to the data to send.	void *
length	The number of bytes to send.	int
flags	Not used.	int
p_dest_addr	A pointer to the structure holding the remote host address.	socket_sockaddr *
dest_len	The length of the *p_dest_addr* structure.	socket_socklen_t

Return values

Return value	Description
The number of bytes sent.	Successful execution.
SOCKET_ERROR	Operation failed.

socket_setopt

Use this function to set the value of the requested socket option.

Format

```
int socket_setopt(
    int                 sockfd,
    int                 level,
    int                 option_name,
    const void *        p_option_value,
    socket_socklen_t    option_len )
```

Arguments

Argument	Description	Type
sockfd	The socket descriptor.	int
level	The socket option level.	int
option_name	The socket option identifier.	int
p_option_value	A pointer to the buffer with the socket option value.	void *
option_len	The length of the socket option data.	socket_socklen_t

Return values

Return value	Description
SOCKET_SUCCESS	Successful execution.
SOCKET_ERROR	Operation failed.

5.3 IPv6 Address Checking Macros

These macros are used in IPv6 systems to check the type of an address. They are defined in section 6.7 of: RFC 2553.

Macro	Description
SOCKET_IN6_IS_ADDR_UNSPECIFIED	Returns a non-zero value if the address is unspecified.
SOCKET_IN6_IS_ADDR_LOOPBACK	Returns a non-zero value if the address is a loopback address.
SOCKET_IN6_IS_ADDR_MULTICAST	Returns a non-zero value if the address is multicast.
SOCKET_IN6_IS_ADDR_LINKLOCAL	Returns a non-zero value if the address is link local, but not an IPv6 multicast link local address.
SOCKET_IN6_IS_ADDR_SITELOCAL	Returns a non-zero value if the address is site local, but not an IPv6 multicast site local address.
SOCKET_IN6_IS_ADDR_V4MAPPED	Returns a non-zero value if the address is IPv4 mapped.
SOCKET_IN6_IS_ADDR_V4COMPAT	Returns a non-zero value if the address is IPv4-compatible.
SOCKET_IN6_IS_ADDR_MC_NODELOCAL	Returns a non-zero value if the address is multicast node local.
SOCKET_IN6_IS_ADDR_MC_LINKLOCAL	Returns a non-zero value if the address is multicast link local.
SOCKET_IN6_IS_ADDR_MC_SITELOCAL	Returns a non-zero value if the address is multicast site local.
SOCKET_IN6_IS_ADDR_MC_ORGLOCAL	Returns a non-zero value if the address is multicast organization local.
SOCKET_IN6_IS_ADDR_MC_GLOBAL	Returns a non-zero value if the address is multicast global.

5.4 Error Codes

The main return values are the following:

Error Code	Value	Meaning
SOCKET_SUCCESS	0	Successful execution.
SOCKET_ERROR	-1	Operation failed.

The following table lists all the error codes that the Sockets API calls can set in the global error field. To check an error code after a function has failed, call **socket_get_errno()**.

Error Code	Meaning
SOCKET_EBADF	Bad file descriptor.
SOCKET_ECONNABORTED *	Software caused connection abort.
SOCKET_ECONNRESET *	The connection was reset by the peer.
SOCKET_EFAULT *	General error.
SOCKET_EINVAL	Invalid argument.
SOCKET_ENETDOWN *	The network is down.
SOCKET_ENOTCONN *	The transport endpoint is not connected.
SOCKET_ETIMEDOUT	The operation timed out.
SOCKET_EAGAIN	Try again.
SOCKET_ENOMEM	No memory available.
SOCKET_EPERM *	The operation is not permitted.
SOCKET_EALREADY	A connection request is already in progress.
SOCKET_ENOTFOUND	Not found.

Note: If one of the errors marked * occurs, the socket cannot be used so close it.

5.5 Types and Definitions

This section describes the main elements (structures, typedefs, and macros) that are used. These are defined in the API header files of this package and the **mip_base** package.

Sockets Macros and Definitions

Socket error value

Return Value	Value
SOCKET_SUCCESS	0
SOCKET_ERROR	-1

Socket domain

Return Value	Description
SOCKET_AF_INET	IPv4
SOCKET_AF_INET6	IPv6

IPv4 address prefix

Return Value	Value	Description
IP_V4_MAP_IP_V6_ADDR_PREFIX_SIZE	12	The size of the prefix in an IP_V4_ENABLE address mapped as an IPv6 address.

Socket types

Return Value	Description
SOCKET_SOCK_STREAM	Stream (TCP) connection.
SOCKET_SOCK_DGRAM	Datagram (UDP) connection.

Any address

Return Value	Description
SOCKET_IN_ADDR_ANY	Zero IP address.

Option levels

Option Level Name	Description
SOCKET_SOL_SOCKET	Sets the option level to Socket options.
SOCKET_IPPROTO_IP	Sets the option level to IP options.

Socket options

Return Value	Description
SOCKET_SO_RCVTIMEO	Used to set a timeout on waiting to receive data on a socket.
SOCKET_SO_SNDTIMEO	Used to set a timeout on waiting to transmit data on a socket.
SOCKET_SO_PRIORITY	Used to set the TOS (DS) field for a socket on all outgoing IP packets. This can be used for SOCKET_SOCK_DGRAM and SOCKET_SOCK_STREAM sockets. The option is an unsigned char that provides the TOS value to use. The current state of the option can be retrieved by using **socket_getopt()**.
SOCKET_SO_LINGER	This is only used for SOCKET_SOCK_STREAM (TCP). When socket_linger is enabled, **socket_close()** waits for transmission of TCP packets to complete before disconnecting to prevent data loss.

Select Macros

Return Value	Description
SOCKET_FD_ISSET(fd, fdset)	Checks whether the specified socket event has occurred on the specified *fdset*.
SOCKET_FD_SET(fd, fdset)	Sets the specified socket in the *fdset*.
SOCKET_FD_CLR(fd, fdset)	Clears the specified socket in the *fdset*.
SOCKET_FD_ZERO(fdset)	Clears the *fdset*.

IP options

Return Value	Description
SOCKET_IP_MULTICAST_TTL	Sets the TTL value to use for outgoing multicast packets on a socket. By default this is 1 (according to the standard). The option is an unsigned char that tells the TTL to use. This option can be retrieved by using **socket_getopt()**.
SOCKET_IP_MULTICAST_IF	Sets the interface to use for sending multicast packets on a socket. The option is "*struct socket_in_addr*" which contains the IP address of the local interface. If address is set to SOCKET_IN_ADDR_ANY, the system automatically chooses the interface (it always uses the interface which has the default gateway). This option can be retrieved by using **socket_getopt()**.
SOCKET_IP_ADD_MEMBERSHIP, SOCKET_IP_DROP_MEMBERSHIP	Add/drop membership of a multicast group. The option is a "*struct socket_ip_mreq*", where: • *imr_multiaddr* is the IP address of the multicast group to subscribe. • *imr_interface* is the local IP address of the interface where membership needs to be added. If *imr_interface* is set to SOCKET_IN_ADDR_ANY the system automatically chooses the interface, which is always the interface with the default gateway. This option cannot be retrieved by using **socket_getopt()**.
SOCKET_SO_KEEPALIVE	For TCP only: enables/disables TCP keep-alive mechanism.
SOCKET_SO_BROADCAST	For UDP only: enables/disables broadcast datagrams.
SOCKET_SO_REUSEADDR	For TCP and UDP: enables/disables SOCKET_SO_REUSEADDR.
SOCKET_SO_NODELAY	For TCP only: enables/disables TCP NO_DELAY.

Sockets ioctl commands

Return Value	Description
SOCKET_FIONBIO	IOCTL to set blocking mode.
SOCKET_FIONREAD	IOCTL to request unread bytes.

Poll events

Event	Description
SOCKET_POLLIN	There is data to read.
SOCKET_POLLPRI	There is urgent data to read (not used).
SOCKET_POLLOUT	The socket has enough send buffer space for writing more data.
SOCKET_POLLERR	Error condition.
SOCKET_POLLHUP	The connection has hung up.
SOCKET_POLLNVAL	Invalid request: fd not open.

Sockets Structure Definitions

socket_sockaddr

Element	Type	Description
sa_family	unsigned short	This is always SOCKET_AF_INET in this implementation.
sa_data[14]	char	Contains specific socket address data.

socket_in_addr

Element	Type	Description
s_addr	unsigned long	The IP address.

socket_in6_addr

Element	Type	Description
s6_addr[16]	uint8_t	The IPv6 address.

socket_sockaddr_in

Element	Type	Description
sin_family	short	This is always SOCKET_AF_INET in this implementation.
sin_port	unsigned short	The port number.
sin_addr	struct socket_in_addr	The IP address.
sin_zero[8]	uint8_t	Reserved.

socket_sockaddr_in6

Element	Type	Description
sin6_family	short	This is always SOCKET_AF_INET6 in this implementation.
sin6_port	unsigned short	The transport layer port number.
sin6_flowinfo	unsigned short	IPv6 flow information.
sin6_addr	struct socket_in_addr	The IPv6 address.
sin6_scope_id	uint32_t	A set of interfaces used for a scope.

socket_ip_mreq

Element	Type	Description
imr_multiaddr	struct socket_in_addr	The multicast IP address of the group.
imr_interface	struct socket_in_addr	The local interface's IP address.

socket_hostent

This is used by the **socket_gethostbyname()** and **socket_gethostbyaddr()** functions. See the function headers in the file **api_ip_socket.h**.

Element	Type	Description
* h_name	char	The host name.
* h_aliases	char *	Alias list.
h_addrtype	int	The host address type.
h_length	int	The length of the address.
* h_addr_list	char *	List of addresses.

t_fd_set

Element	Type	Description
fdmask[((SOCK_TABLE_SIZE * 2U) + 31U) / 32U]	uint32_t	

t_pollfd

Element	Type	Description
fd	int	The socket descriptor.
events	unsigned short	Events to wait for.
revents	unsigned short	Received events.

socket_linger

Element	Type	Description
l_onoff	int	Set this to 0 to disable lingering. Any other value enables it.
l_linger	int	The number of seconds to linger for; 0 means wait forever.

socket_timeval

Element	Type	Description
tv_sec	long	Seconds.
tv_usec	long	Microseconds.

Note: If both values are -1 this means forever.

IPv6 loopback address

```
#define SOCKET_IN6ADDR_LOOPBACK_INIT { { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 } } }
```

IPv6 wild card address

```
#define SOCKET_IN6ADDR_ANY_INIT { { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } } }
```

6 Integration

This section describes all aspects of the Sockets module that require integration with your target project.

This includes porting and configuration of external resources.

6.1 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

This module uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	0

6.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

The Sockets module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_memcmp()	psp_base	psp_string	Compares two blocks of memory.
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.
psp_strncpy()	psp_base	psp_string	Copies one string of defined length to another.
psp_strnlen()	psp_base	psp_string	Gets the length of a string.

The Sockets module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.
PSP_WR_BE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as big-endian to a memory location.