SNTP Client User Guide

Version 1.50

For use with Simple Network Time Protocol (SNTP) Client module versions 1.08 and above

Date: 14-Jun-2017 17:27

Table of Contents

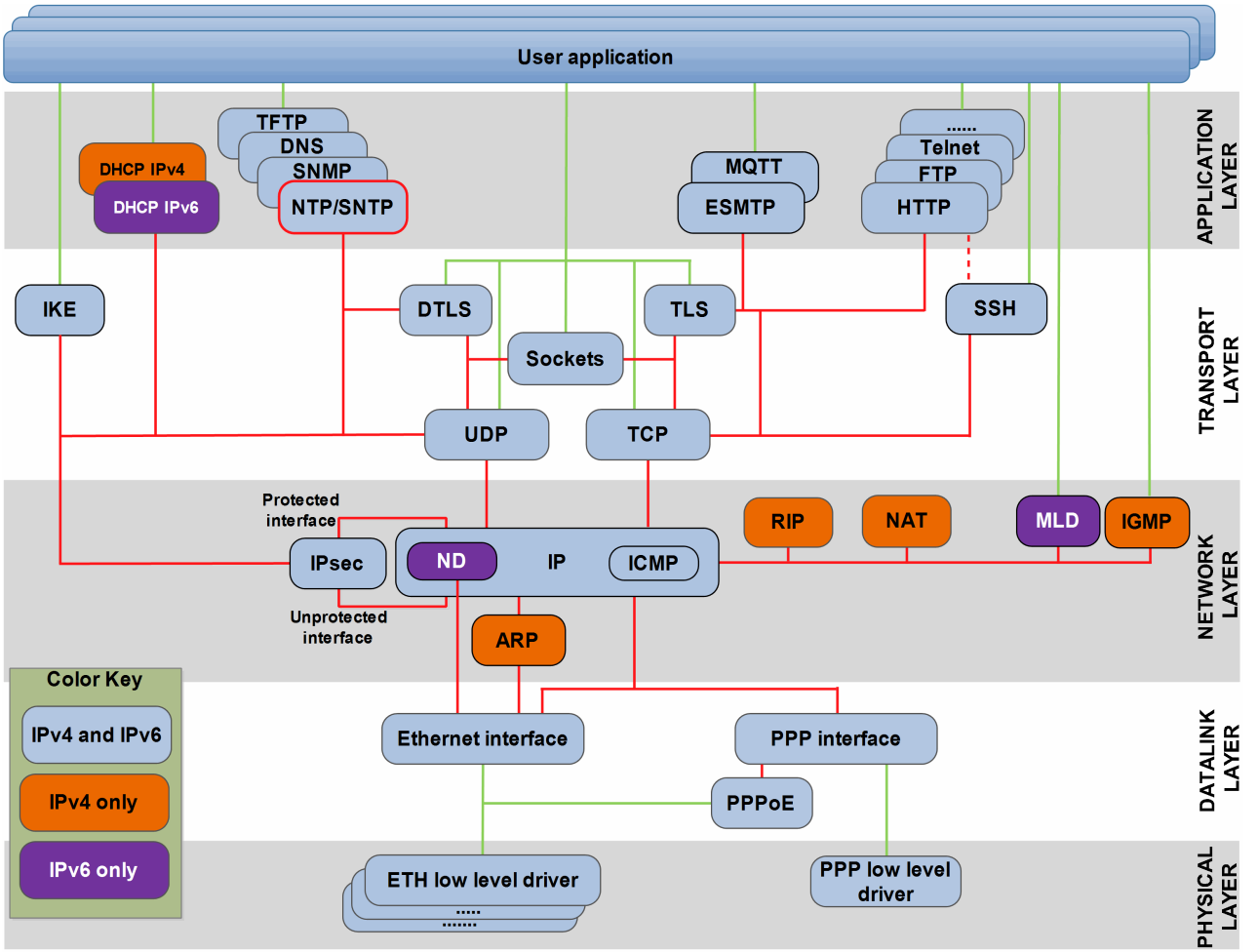
1 System Overview

1.1 Introduction

This guide is for those who want to implement HCC Embedded's Simple Network Time Protocol (SNTP) client module. This client module comprises a single task which handles communication with SNTP servers.

SNTP is used to synchronize clocks on computer systems in packet-switched networks. SNTP includes the On-Wire Protocol.

The SNTP 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.)



Note: This diagram shows both SNTP and its more complex alternative, the Network Time Protocol (NTP). These are mutually exclusive.

1.2 Time Accuracy

In general the time accuracy, the difference between the client time and the server time, depends on two factors:

- The transmission time between client and server; this is affected by network traffic. With SNTP there is only one server.
- The precision of the client's clock.

Transmission time

Assuming that:

- T1 is the time needed to transmit a packet from client to server.
- T2 is the time needed to transmit a packet from server to client.

Then the time difference between the client and server time is equal to T2-T1. For example, if time T2-T1 is 1 ms, the client clock has a 1 ms offset from the server clock. This means that time accuracy is greatest when the transmission times are equal.

Local clock precision

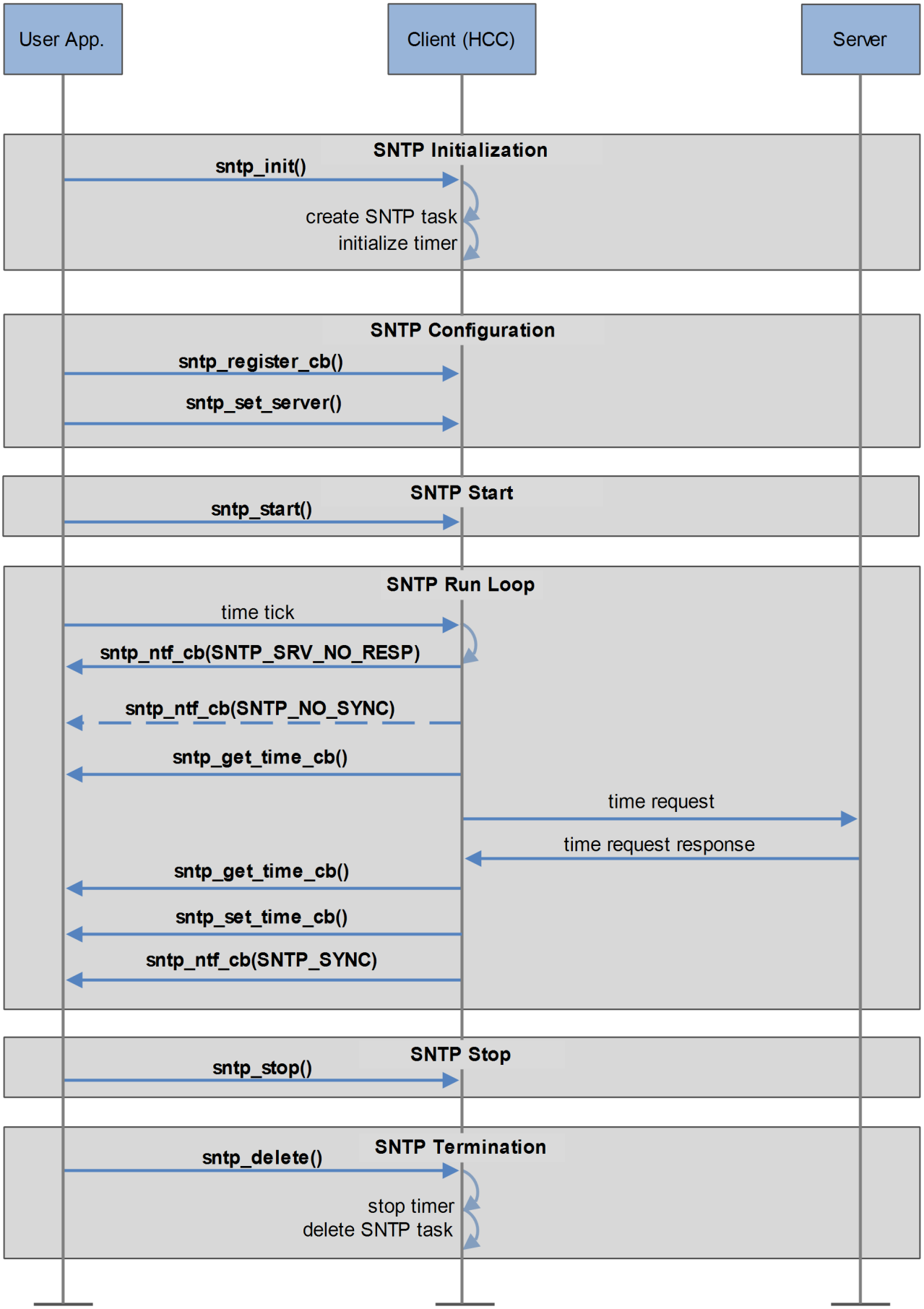
The time accuracy cannot be greater than the local clock precision. SNTP continuously corrects the time offset.

1.3 Overview of Operation

The SNTP module uses time servers on the internet to synchronize its own clock.

The following time sequence diagram shows how the module is used and specifically:

1. Module initialization.
2. How the module runs and interacts with SNTP servers.
3. How the module uses user-defined callback functions to update the system time.
4. Module termination.


User App.
Client (HCC)
Server
SNTP Initialization
sntp_init()
create SNTP task
initialize timer
SNTP Configuration
sntp_register_cb()
sntp_set_server()
SNTP Start
sntp_start()
SNTP Run Loop
time tick
sntp_ntf_cb(SNTP_SRV_NO_RESP)
sntp_ntf_cb(SNTP_NO_SYNC)
sntp_get_time_cb()
time request
time request response
sntp_get_time_cb()
sntp_set_time_cb()
sntp_ntf_cb(SNTP_SYNC)
SNTP Stop
sntp_stop()
SNTP Termination
sntp_delete()
stop timer
delete SNTP task

1.4 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Complies with HCC's MISRA-compliant TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- The SNTP implementation is compliant with RFC 1769, RFC 2030, and RFC 4330.
- Supports the the On-Wire Protocol (RFC 5905, Chapter 8).
- Supports SNTP unicast (RFC 1769), SNTP broadcast (RFC 2030), and SNTP manycast (RFC 4330).

1.5 Packages and Documents

Packages

The following table 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_app_sntp	The SNTP client package (this package).
mip_base	The TCP/IP Dual Stack base package.
ip_base_v4, ip_base_v6	The base packages for IPv4 and IPv6, respectively.
mip_udp	The UDP 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 UDP User Guide

This manual documents the UDP package.

HCC SNTP Client User Guide

This is this document.

1.6 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: SNTP Client User Guide.
- For the history of changes made to the package code itself, see History: ip_app_sntp.

The current version of this manual is 1.50. The previous versions are as follows:

Manual version	Date	Software version	Reason for change
1.50	2017/06/14	1.08	New *Change History* format.
1.40	2017/03/28	1.06	Updated network diagram. *Added Change History*.
1.30	2017/01/16	1.04	Updated network diagram.
1.20	2015/11/02	1.01	Added *Overview of Operation*.
1.10	2015/09/04	1.01	Small changes.
1.00	2015/05/08	1.01	First release.

2 Source File List

The following sections describe 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_app_sntp.h** is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_sntp.h** is the only file in the module that you should modify. Configure this as required. For details of the options, see Configuration Options.

2.3 System File

The file **src/ip/apps/sntp/sntp.c** is the source code file. **This file should only be modified by HCC**.

2.4 Version

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

3 Configuration Options

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

SNTP_CLIENT_TASK_STACK_SIZE

The size of the stack used by the client task. The default is 512.

SNTP_STATIC_SRV_ADDR

Keep this at the default of 1 to prevent a user changing the server's addresses dynamically. Otherwise, set it to 0.

SNTP_CLIENT_PORT

The SNTP client port. The default is 1023; do not set a value lower than this.

SNTP_POLL_INTERVAL

The time interval used for communicating with the SNTP server. The default is 4.

SNTP_USE_SOCKET

Keep this at the default of 0 to use the native HCC implementation. Set it to 1 to use the BSD sockets implementation.

SNTP_USE_IP_V6_SOCKET

Keep this at the default of 0 to use the BSD sockets implementation with IPv4. Set it to 1 to use BSD sockets with IPv6.

SNTP_USE_STD_SOCKET

Keep this at the default of 0 to use the HCC socket implementation. Set it to 1 to use the standard socket implementation.

4 Application Programming Interface

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

4.1 Module Management

The functions are the following:

Function	Description
sntp_init()	Initializes the module and allocates the required resources.
sntp_start()	Starts the module.
sntp_stop()	Stops the module.
sntp_delete()	Deletes the module and releases the resources it used.

sntp_init

Use this function to initialize the client module and allocate the required resources.

> **Note:** Call this before any other SNTP function.

Format

```
t_sntp_ret sntp_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

sntp_start

Use this function to start the client module.

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

Format

```
t_sntp_ret sntp_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

sntp_stop

Use this function to stop the client module.

This clears connection data and closes all open SNTP connections.

Format

```
t_sntp_ret sntp_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

sntp_delete

Use this function to delete the client module and release the associated resources.

Format

```
t_sntp_ret sntp_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

4.2 Client Functions

The functions are the following:

Function	Description
sntp_set_server()	Sets the SNTP server.
sntp_register_cb()	Registers the time synchronization interface callback functions, if these are used, with the SNTP module.

sntp_set_server

Use this function to set the SNTP server configuration.

Note:

- This function can only be called in the stopped state. The system is in this state between calls of **sntp_init()** and **sntp_start()**, and also after a call of **sntp_stop()**.
- If the option SNTP_STATIC_SRV_ADDR is set to a value other than zero, this function is not available in order to save memory; a predefined pool of static addresses is used instead.

Format

```
t_sntp_ret sntp_set_server (
    uint8_t       b_sec,
    t_ip_port *   p_port,
    uint8_t       ver,
    uint8_t       mode )
```

Arguments

Argument	Description	Type
b_sec	Set this to TRUE if the secondary server setting should be changed.	uint8_t
p_port	A pointer to the server IP address and port number.	t_ip_port *
ver	The SNTP protocol version to use when communicating with the server.	uint8_t
mode	The client operating mode (SNTP_MODE_*).	uint8_t

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_PARAM_ERR	A parameter passed to this function is invalid.

sntp_register_cb

Use this function to register the time synchronization interface callback functions, if these are used, with the SNTP module.

> **Note:** This function can only be called in the stopped state. The system is in this state between calls of **sntp_init()** and **sntp_start()**, and also after a call of **sntp_stop()**.

Format

```
t_sntp_ret sntp_register_cb ( const t_sntp_cb_dsc * const p_cb_dsc )
```

Arguments

Argument	Description	Type
p_cb_dsc	A pointer to the time interface descriptor. This structure describes the callback functions to the time interface.	t_sntp_cb_dsc *

Return Values

Return value	Description
SNTP_SUCCESS	Successful execution.
SNTP_PARAM_ERROR	One of the callback function pointers is NULL.

4.3 Callback Functions

These functions provide the interface to the SNTP server. You can implement these as required.

Note:

- It is the user's responsibility to provide any callback functions the application requires. Providing such functions is optional.
- All callback functions are called from the same task context and are protected against concurrent calls; no callback function can interrupt another callback function.

The callbacks are the following:

Function	Description
t_sntp_get_time_cb()	Specifies the format of a callback function you may call to obtain a new time after synchronization with the SNTP server.
t_sntp_set_time_cb()	Specifies the format of a callback function you may call to set a new system time after synchronization with the SNTP server.
t_sntp_ntf_cb()	Specifies the format of a callback function you may call to notify a user about SNTP events.

t_sntp_get_time_cb

The **t_sntp_get_time_cb()** definition specifies the format of a callback function that may be called to get the current system time after synchronization with the SNTP server.

SNTP 64 bit timestamps comprise a 32 bit part for seconds and a 32 bit part for fractional seconds. The returned parameters contain these two components.

Format

```
typedef void ( * t_sntp_get_time_cb ) (
    uint32_t *   p_seconds,
    uint32_t *   p_fraction )
```

Arguments

Argument	Description	Type
p_seconds	A pointer to the number of seconds.	uint32_t *
p_fraction	A pointer to the number of fractional seconds.	uint32_t *

Return Codes

Code	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

t_sntp_set_time_cb

The **t_sntp_set_time_cb()** definition specifies the format of the callback function which may be called to set a new time after synchronization with the SNTP server.

SNTP 64 bit timestamps comprise a 32 bit part for seconds and a 32 bit part for fractional seconds. The parameters contain these two components.

Format

```
typedef void ( * t_sntp_set_time_cb ) (
    uint32_t   seconds,
    uint32_t   fraction )
```

Arguments

Argument	Description	Type
seconds	The number of seconds.	uint32_t
fraction	The number of fractional seconds.	uint32_t

Return Codes

Code	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

t_sntp_ntf_cb

The **t_sntp_ntf_cb()** definition specifies the format of a callback function which may be used to notify a user about SNTP events.

Format

```
typedef void ( * t_sntp_ntf_cb ) ( uint32_t ntf )
```

Arguments

Argument	Description	Type
ntf	The notification type.	uint32_t

Return Codes

Code	Description
SNTP_SUCCESS	Successful execution.
SNTP_ERROR	Operation failed.

4.4 Error Codes

If a function executes successfully, it returns with SNTP_SUCCESS, a value of 0. The following table shows the meaning of the SNTP error codes.

> **Note:** Also check error code values in the base system by using the relevant *HCC TCP/IP Stack System User Guide*.

Return Value	Value	Description
SNTP_SUCCESS	0	Successful execution.
SNTP_ERROR	1	Operation failed.
SNTP_PARAM_ERR	2	A parameter passed to this function is invalid.

4.5 Types and Definitions

Notification Types

The following notifications may be sent by an SNTP client:

Element	Value	Description
SNTP_NTF_NO_SYNCH	0U	Time is not synchronized with server.
SNTP_NTF_SYNCH	1U	Time is synchronized with server.
SNTP_NTF_SRV_NO_RESP	2U	Failed to get time from server.

Client Operating Modes

The following client operating modes may be used:

Element	Value	Description
SNTP_MODE_UNICAST	0U	Unicast mode.
SNTP_MODE_BROADCAST	1U	Broadcast mode.
SNTP_MODE_MANYCAST	2U	Manycast mode.

SNTP Protocol Version

The following protocol versions may be used:

Element	Value	Description
SNTP_HDR_VN_1	0x08U	SNTP protocol version 1.
SNTP_HDR_VN_2	0x10U	SNTP protocol version 2.
SNTP_HDR_VN_3	0x18U	SNTP protocol version 3.
SNTP_HDR_VN_4	0x20U	SNTP protocol version 4.

t_sntp_cb_dsc

The *t_sntp_cb_dsc* structure defines the time synchronization interface.

Its elements are as follows:

Element	Type	Description
sntpc_get_time_cb	t_sntp_get_time_cb	The get current time callback.
sntpc_set_time_cb	t_sntp_set_time_cb	The set current time callback.
sntpc_ntf_cb	t_sntp_ntf_cb	The notify callback.

5 Integration

This section describes all aspects of the SNTP Client module that require integration with your target project. This includes porting and configuration of external resources.

5.1 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL). This allows modules 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	1
Mutexes	1
Events	1

5.2 Utilities

The code creates and uses a single timer in the **hcc_timer** module.

The **hcc_timer** module is included in your system when you install the base TCP/IP modules.

5.3 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 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_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_strncmp()	psp_base	psp_string	Compares two strings of defined length.

The 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_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-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.