NTP Client User Guide

Version 1.80

For use with Network Time Protocol (NTP) Client module versions 1.17 and above

Date: 13-Nov-2017 15:08

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

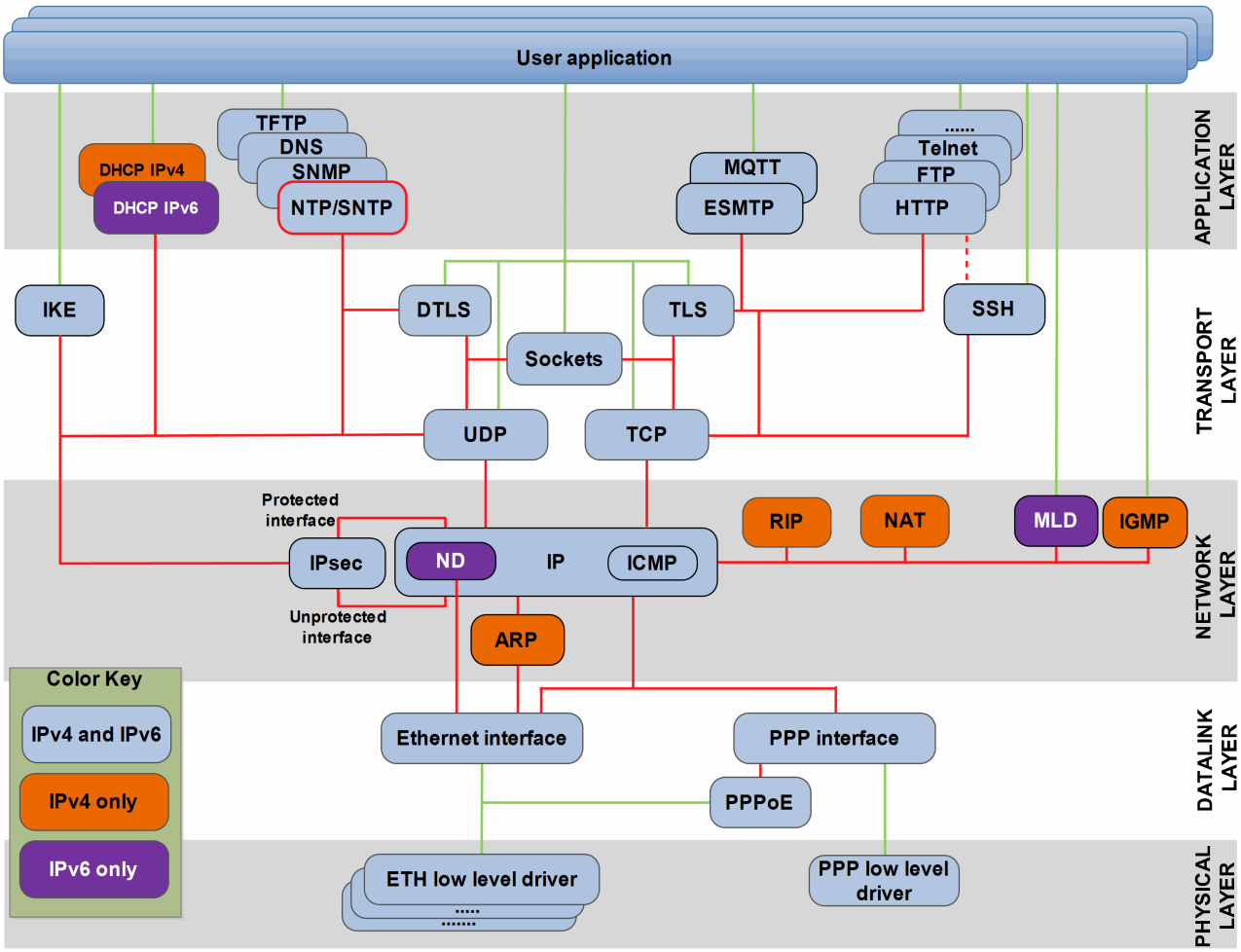
- Introduction – describes the main elements of the module. This section includes a diagram showing the position of this module within HCC's TCP/IP stack.
- Feature Check – summarizes the main features of the module as bullet points.
- Overview of Operation – presents a time sequence diagram that shows how the module operates.
- Time Accuracy – describes the two factors that influence the difference between the client time and the server time.
- 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 implement HCC Embedded's Network Time Protocol (NTP) client module. This client module comprises a single task which handles communication with NTP servers.

NTP is used to synchronize clocks on computer systems in packet-switched networks. An NTP client implements mitigation algorithms and authentication. NTP authentication uses keys with the Message Digest algorithm 5 (MD5).

The NTP module is part of HCC's 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 NTP and its alternative, the Simple Network Time Protocol (SNTP). These are mutually exclusive.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Compliant with HCC's MISRA-compliant TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compliant with RFC 5905. It supports the the On-Wire Protocol (RFC 5905, Chapter 8).
- Backward-compatible with RFC 1305.
- Supports NTP unicast (RFC 1769), NTP broadcast (RFC 2030), and NTP manycast (RFC 4330).
- Supports the NTP Authentication symmetric key (MD5; RFC 1305, Appendix C).
- User demo package is available.

The following NTP algorithms (RFC 5905) are supported:

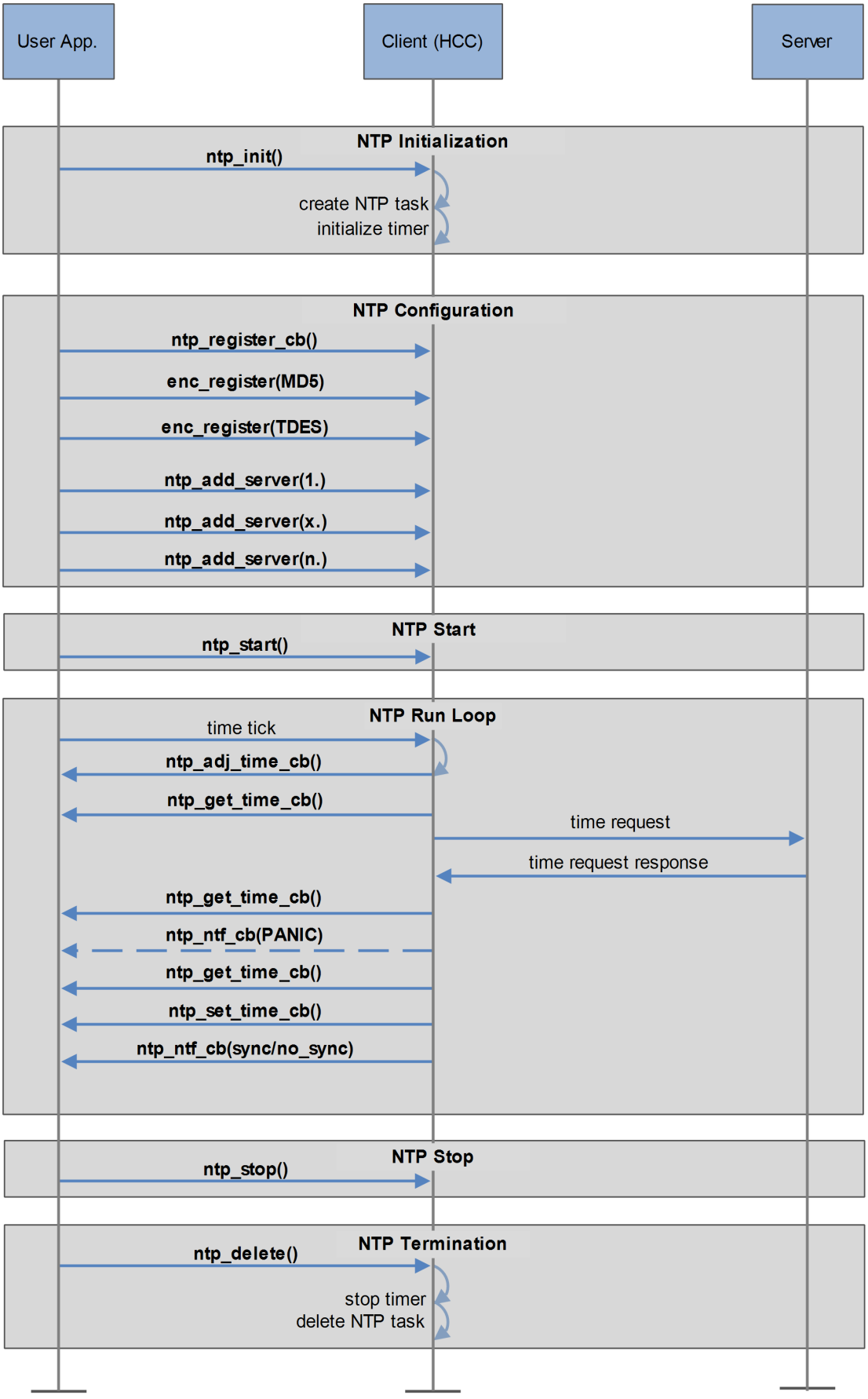
- Clock filter algorithm.
- Selection algorithms.
- Cluster algorithms.
- Combine algorithms.
- Clock Discipline Algorithm (CDA).
- Clock-adjust process.

1.3 Overview of Operation

The NTP 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 NTP servers.
3. How the module uses user-defined callback functions to update the system time.
4. Module termination.


User App.
Client (HCC)
Server
NTP Initialization
ntp_init()
create NTP task
initialize timer
NTP Configuration
ntp_register_cb()
enc_register(MD5)
enc_register(TDES)
ntp_add_server(1.)
ntp_add_server(x.)
ntp_add_server(n.)
NTP Start
ntp_start()
NTP Run Loop
time tick
ntp_adj_time_cb()
ntp_get_time_cb()
time request
time request response
ntp_get_time_cb()
ntp_ntf_cb(PANIC)
ntp_get_time_cb()
ntp_set_time_cb()
ntp_ntf_cb(sync/no_sync)
NTP Stop
ntp_stop()
NTP Termination
ntp_delete()
stop timer
delete NTP task

1.4 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. In NTP multiple servers are used and those that have the smallest transmission delays are chosen.
- 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. The frequency jitter of the local clock is corrected but NTP needs 15 minutes to go into frequency adjustment state.

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_ntp	The NTP client package (this package).
ip_base_v4	The TCP/IP Stack IPv4 package.
mip_udp	The UDP package, needed if NTP is used with HCC's native TCP/IP.
ip_socket	The IP Sockets package, needed if NTP is used with BSD sockets.
psp_template_socket	A wrapper to connect HCC modules that use sockets with the BSD-compliant socket implementation.
psp_template_math	The mathematics Platform Support Package (PSP).
enc_base	The encryption base package.
ip_app_ntp_demo	The user demo for NTP; this is not required but may be useful.

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 document describes the UDP package.

HCC Sockets Interface User Guide

This document describes the Sockets package.

HCC NTP Client User Guide

This is this document.

1.6 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_app_ntp.

The current version of this manual is 1.80. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.80	2017-11-13	1.17	Removed **NTP_STATIC_SRV_ADDR** option. Removed **psp_math** functions from *PSP Porting*.
1.70	2017-09-05	1.12	Corrected *Packages* list.
1.60	2017-06-20	1.12	New *Change History* format.
1.50	2017-03-28	1.09	Updated network diagram.
1.40	2017-01-16	1.06	Updated network diagram. Added function group tables.
1.30	2016-07-14	1.04	Added software *Change History*. Extended *Introduction*.
1.20	2015-11-02	1.02	Small changes.
1.10	2015-10-27	1.02	Reorganized *System Overview* section.
1.00	2015-05-08	1.02	First online version.

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 and PSP files.

2.1 API Header File

The file **src/api/api_ip_app_ntp.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_ip_app_ntp.h** contains all the configurable NTP parameters. Configure these as required. See Configuration Options for details.

2.3 System Files

The following files are in the directory **src/ip/apps/ntp**. **These files should only be modified by HCC**.

File	Description
ntp.c	Functions source code.
ntp.h	Header file.
ntp_alg.c	Clock and polling functions source code.
ntp_alg.h	Clock and polling functions header file.
ntp_auth.c	Authorization functions source code.
ntp_auth.h	Authorization functions header file.

2.4 Version

The file **src/version/ver_ip_app_ntp.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.

2.5 Platform Support Package (PSP) Files

These files in **src/psp** provide the clock functions and math functions described in PSP Porting.

> **Note:** These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board. See PSP Porting for details.

File	Description
include/psp_math.h	Math functions header file.
include/psp_rtc.h	Real time clock functions header file.
target/psp_math.c	Math functions source code.
target/psp_rtc.c	Real time clock functions source code.

3 Configuration Options

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

> **Note:** Where stated, options use f64p32 format (using 32 bits to represent the fractional part).

3.1 NTP Task Configuration

NTP_CLIENT_TASK_STACK_SIZE

The size of the stack used by the module. The default is 512.

NTP_CLIENT_PORT

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

3.2 General Configuration

These options control NTP client configuration.

NTP_PRECISION

The system time precision. This is an exponent; the value is calculated as $2^{NTP\_PRECISION}$. The default is 32-10.

NTP_MAX_PEERS

The maximum number of peers. The default is 6.

NTP_MAX_SRV

The number of servers in the configuration. The default is 6.

NTP_MAX_DISP

The maximum dispersion time. This is the maximum error of the local clock relative to the reference clock. This is 16s in f64p32 format. The default is (0x1000000000).

NTP_MIN_DISP

The minimum dispersion time. This is 0.01s in f64p32 format. The default is (0x028F5C28).

NTP_MAX_DIST

The maximum synchronization distance. This is 1s in f64p32 format. The default is (0x100000000).

NTP_MAX_STRAT

The maximum stratum number. The default is 16.

NTP_MIN_POLL

The minimum poll interval. This is an exponent; the minimum interval is 2^n seconds. The default is 3.

NTP_MAX_POLL

The maximum poll interval. This is an exponent; the maximum interval is 2^n seconds. The default is 12.

NTP_NSANE

The minimum number of intersection survivors. The default is 3.

NTP_NMIN

The minimum number of cluster survivors. The default is 2.

NTP_SGATE

The spike gate used in the clock filter. The default is 3.

3.3 Clock Discipline Algorithm (CDA) Options

NTP_PANICT

The threshold used to determine when the time difference between host and server is too large. This is termed the "panic threshold". The default is (0x10003E800000000).

NTP_CLKD_STEPT

The step threshold (0.128s) in f64p32 format. The default is (0x20C49BA5).

NTP_CLKD_WATCH

The stepout threshold in seconds. The default is (900*0x100000000)

NTP_CLKD_PLL_SHIFT

The PLL loop gain shift value. This is an exponent value; the gain is $2^{NTP\_CLKD\_PLL\_SHIFT}$. The default is 4.

NTP_CLKD_FLL

The FLL loop gain. The default is (NTP_MAX_POLL + 1).

NTP_CLKD_AVG

The CDA averaging constant. The default is 4.

NTP_CLKD_ALLAN_SHIFT

The compromise Allan intercept value in seconds. This is an exponent; it should be equal to log2 (NTP_CLKD_ALLAN). The default is 10.

NTP_CLKD_ALLAN

The compromise Allan intercept value in f64p32 format. The default is (1500*0x100000000).

NTP_CLKD_LIMIT

The poll-adjust limit. The default is 30.

NTP_CLKD_MAXFREQ

The frequency tolerance (500 ppm). The default is (0x20C49B).

NTP_CLKD_PGATE

The poll-adjust gate value. The default is 4.

NTP_UNREACH_MAX

The number of polls to increase the poll value by in case of an unreachable server. The default is 12.

NTP_BROADCAST_DELAY

The broadcast delay in f64p32 format (0.004s). The default is (0x10624DD).

NTP_LONG_JUMP_ENABLE

Keep the default of 1 to enable a long time jump when synchronizing for the first time. This suppresses the PANIC notify when synchronizing to a server whose time is significantly different to the device's system time.

Note: If NTP_LONG_JUMP_ENABLE is not set, NTP may never synchronize if the system clock time is much different from that of NTP servers.

3.4 Implementation Options

NTP_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.

NTP_USE_IP_V6_SOCKET

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

NTP_USE_STD_SOCKET

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

NTP_SECURE_ENABLE

Keep this at the default of 0 to disable the NTP authorization mechanism. Set it to 1 to use the authorization mechanism.

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
ntp_init()	Initializes the module and allocates the required resources.
ntp_start()	Starts the module.
ntp_stop()	Stops the module.
ntp_delete()	Deletes the module and releases the resources it used.

ntp_init

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

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

Format

```
t_ntp_ret ntp_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

ntp_start

Use this function to start the client module.

If one of the configured servers uses a key identifier that is not in the list of authorization keys, this function returns NTP_KEY_MISSING. If the user configures less than NTP_NSANE NTP servers, the function returns NTP_NOT_EN_SERVERS.

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

Format

```
t_ntp_ret ntp_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.
NTP_NOT_EN_SERVERS	Too few servers are configured to allow the mitigation algorithm to pass.

ntp_stop

Use this function to stop the client module.

This clears connection data and closes all open NTP connections.

Format

```
t_ntp_ret ntp_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

ntp_delete

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

Format

```
t_ntp_ret ntp_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

4.2 Client Functions

The functions are the following:

Function	Description
ntp_add_server()	Adds the NTP server configuration to the NTP server list.
ntp_del_server()	Deletes an NTP server configuration from the NTP servers list.
ntp_register_md5()	Registers the MD5 hash algorithm handle with the NTP module.
ntp_register_cb()	Registers the time synchronization interface callback functions, if these are used, with the NTP module.

ntp_add_server

Use this function to add the NTP server configuration to the NTP server list.

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

Format

```
t_ntp_ret ntp_add_server (
    t_ip_port *   p_port,
    uint8_t       ver,
    uint8_t       mode,
    t_ntp_key *   p_key,
    uint8_t *     p_id )
```

Arguments

Argument	Description	Type
p_port	A pointer to the server IP address and port number.	t_ip_port *
ver	The NTP protocol version to use when communicating with the server.	uint8_t
mode	The client operating mode (NTP_MODE_*).	uint8_t
p_key	A pointer to a structure containing key data to be used for authorization. If authorization is disabled, this parameter is discarded. Key identifier 0 is reserved by the NTP protocol.	t_ntp_key *
p_id	On return, a pointer to the server configuration's identifier.	uint8_t

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_PARAM_ERR	A parameter passed to this function is invalid. This is returned if NTP_FLAGS_AUTH_NEEDED is set and *p_key* is NULL.
NTP_NO_FREE_SPACE	Cannot add the server/key because there is no free slot in the configuration table.
NTP_ERROR	The function was called when not in the stopped state.

ntp_del_server

Use this function to delete an NTP server configuration from the NTP servers list.

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

Format

```
t_ntp_ret ntp_del_server ( uint8_t p_id )
```

Arguments

Argument	Description	Type
p_id	The server configuration's identifier.	uint8_t

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_PARAM_ERR	The parameter passed to this function is invalid.
NTP_ERROR	The function was called when not in the stopped state.

ntp_register_md5

Use this function to register the MD5 hash algorithm handle with the NTP module.

This handle is obtained from the Embedded Encryption Manager.

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

Format

```
t_ntp_ret ntp_register_md5( t_enc_ifc_hdl hdl )
```

Arguments

Argument	Description	Type
hdl	The handle of the MD5 driver.	t_enc_ifc_hdl

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_PARAM_ERR	The handle is already registered.
NTP_ERROR	The function was called when not in the stopped state.

ntp_register_cb

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

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

Format

```
t_ntp_ret ntp_register_cb ( const t_ntp_cb_dsc * const p_cb )
```

Arguments

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

Return Values

Return value	Description
NTP_SUCCESS	Successful execution.
NTP_PARAM_ERROR	One of the callback function pointers is NULL.
NTP_ERROR	The function was called when not in the stopped state.

4.3 Callback Functions

These functions provide the interface to the NTP 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_ntp_get_time_cb()	Specifies the format of a callback function you may call to obtain a new time after synchronization with the NTP server.
t_ntp_set_time_cb()	Specifies the format of a callback function you may call to set a new system time after synchronization with the NTP server.
t_ntp_adj_time_cb()	Specifies the format of a callback function you may call to adjust the system time clock frequency.
t_ntp_ntf_cb()	Specifies the format of a callback function you may call to notify a user about NTP events.

t_ntp_get_time_cb

The **t_ntp_get_time_cb()** definition specifies the format of a callback function you may call to obtain a new time after synchronization with the NTP server.

NTP 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_ntp_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
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

t_ntp_set_time_cb

The **t_ntp_set_time_cb()** definition specifies the format of the callback function you may call to set a new system time after synchronization with the NTP server.

NTP 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_ntp_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
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

t_ntp_adj_time_cb

The **t_ntp_adj_time_cb()** definition specifies the format of the callback function you may call to adjust the system time clock frequency.

NTP 64 bit timestamps comprise a 32 bit part for seconds and a 32 bit part for fractional seconds. The *delta* parameter contains these two components.

Format

```
typedef void ( * t_ntp_adj_time_cb ) ( int64_t delta )
```

Arguments

Argument	Description	Type
delta	The time to adjust the system clock by (32 bit seconds, 32 fractional bits).	int64_t

Return Codes

Code	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

t_ntp_ntf_cb

The **t_ntp_ntf_cb()** definition specifies the format of a callback function you may call to notify a user about NTP events.

Format

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

Arguments

Argument	Description	Type
ntf	The notification type.	uint32_t

Return Codes

Code	Description
NTP_SUCCESS	Successful execution.
NTP_ERROR	Operation failed.

4.4 Error Codes

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

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

Return Value	Value	Description
NTP_SUCCESS	0	Successful execution.
NTP_ERROR	1	Operation failed.
NTP_PARAM_ERR	2	A parameter passed to this function is invalid.
NTP_NO_FREE_SPACE	3	Cannot add the server/key because there is no free slot in the configuration table.
NTP_NOT_EN_SERVERS	4	Too few servers are configured to allow the mitigation algorithm to pass.

4.5 Types and Definitions

This section describes the main elements that are defined in the API Header file.

Notification Types

The following notifications may be sent by the client:

Element	Value	Description
NTP_NTF_NO_SYNCH	0U	The time is not synchronised with the server.
NTP_NTF_SYNCH	1U	The time is synchronised with the server.
NTP_NTF_SRV_NO_RESP	2U	An attempt to get the time from the server failed.
NTP_NTF_PANIC	3U	Time unsynchronization is larger than NTP_PANICT. You must set the system clock manually.

Client Operating Modes

The following client operating modes may be used:

Element	Value	Description
NTP_MODE_UNICAST	0U	Unicast mode.
NTP_MODE_BROADCAST	1U	Broadcast mode.
NTP_MODE_MANYCAST	2U	Manycast mode.

Flags

The following flags may be used:

Element	Value	Description
NTP_FLAGS_BURST_EN	4U	Enables burst mode on every poll to a server.
NTP_FLAGS_INIT_BURST_EN	8U	Enables burst mode on connection initialization.
NTP_FLAGS_AUTH_NEEDED	10U	Shows that this server requires authorization.

NTP Protocol Versions

The following protocol versions may be used:

Element	Value	Description
NTP_HDR_VN_1	0x08U	NTP protocol version 1.
NTP_HDR_VN_2	0x10U	NTP protocol version 2.
NTP_HDR_VN_3	0x18U	NTP protocol version 3.
NTP_HDR_VN_4	0x20U	NTP protocol version 4.

t_ntp_cb_dsc

The *t_ntp_cb_dsc* structure defines the time synchronization interface.

Its elements are as follows:

Element	Type	Description
ntpc_get_time_cb	t_ntp_get_time_cb	The get current time callback.
ntpc_set_time_cb	t_ntp_set_time_cb	The set current time callback.
ntpc_adj_time_cb	t_ntp_adj_time_cb	The adjust time callback.
ntpc_ntf_cb	t_ntp_ntf_cb	The notify callback.

t_ntp_key

The *t_ntp_key* structure describes the key used for authentication.

Its elements are as follows:

Element	Type	Description
ntpk_id	uint16_t	The authentication identifier that is used during communication with the server. Value 0 is reserved.
ntpk_type	uint8_t	The key type: 1 = MD5.
ntpk_key [NTP_KEY_LENGTH]	uint8_t	Each key is constructed as in the Berkeley Unix distributions.

t_ntp_srv_conf

The *t_ntp_srv_conf* structure describes port settings that are used to connect to the NTP server.

Its elements are as follows:

Element	Type	Description
ntpk_id	t_ip_port	The NTP server address and IP port.
ntsc_ver	uint8_t	The NTP protocol version.
ntsc_mode	uint8_t	The NTP client operating mode.
ntsc_key	uint16_t	The key authorization identifier.

5 Integration

This section describes all aspects of the NTP 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_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.

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.