

# HS NTP Library v1.2 User Manual

Revision: 1.2

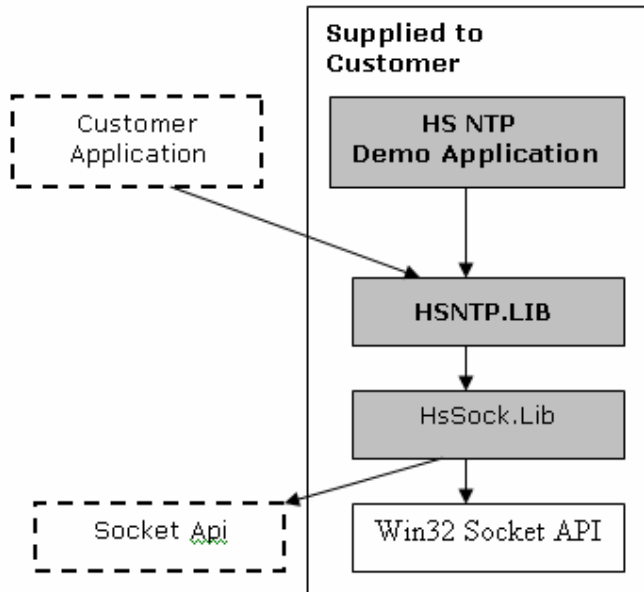
Date: 05 November 2009

<b>HS NTP LIBRARY V1.2 USER MANUAL</b>	<b>1</b>
<b>REVISION: 1.2</b>	<b>1</b>
<b>DATE: 05 NOVEMBER 2009</b>	<b>1</b>
<b>1 INTRODUCTION</b>	<b>2</b>
<b>2 HS NTP API</b>	<b>3</b>
<b>2.1 HSNTPINIT</b>	<b>3</b>
2.1.1 INITIALISATION STRUCTURE DEFINITION (HS_NTP_API_T)	3
<b>2.2 HSNTPDESTROY</b>	<b>4</b>
<b>2.3 HSNTPGETERRSTR</b>	<b>4</b>
<b>2.4 HSNTPGETIME</b>	<b>5</b>
<b>3 HS NTP TO USER EVENT CALLBACK AND EVENTS</b>	<b>6</b>
<b>3.1 EVENT CALLBACK PROTOTYPE</b>	<b>6</b>
<b>3.2 EVENTS</b>	<b>6</b>
3.2.1 NTP TIME REPLY STRUCTURE (HS_NTP_INFO_T)	7

## 1 Introduction

HS NTP is a software library in C (supplied with full source code) which implements the client side of Network Time Protocol (NTP) over UDP socket layer according to RFC 1769 and RFC1305. The library allows the user application to synchronize local time with the time of remote NTP server.

The library is a stand-alone module which links directly to customer application:



HS NTP Library incorporates the necessary server response processing required to comply to a simple implementation of NTP client.

## 2 HS NTP API

### 2.1 HsNtpInit

```
int HsNtpInit (hs_ntp_api_t *api);
```

#### DESCRIPTION

*This function initialises HS NTP Library. It must be called at initialisation, before any other functions are called. Calling HsNtpInit twice will return an error. You can call HsNtpDestroy first to deinitialise HS NTP Library and then call HsNtpInit again*

#### PARAMETERS

Type	Name	Description
hs_ntp_api_t	*api	Pointer to parameter structure, please see next section for details

#### RETURNS

- *HS\_NTP\_RC\_ALRINIT – HS NTP Library already initialised*
- *HS\_NTP\_RC\_INV\_PAR – invalid parameters specified*
- *HS\_NTP\_RC SOCKINIT\_FAIL – socket layer initialisation failed*
- *HS\_NTP\_RC\_OK - Success*

#### 2.1.1 Initialisation Structure Definition (hs\_ntp\_api\_t)

Field name	Description
start_timer_t    *start_timer;	<p>Pointer function in user code used by HS NTP to start a timer</p> <p><u>Prototype:</u> long start_timer_t(unsigned long secs, timer_callback_t *callback);</p> <p><u>Parameters:</u> secs – timeout in seconds *callback – pointer to function in HS NTP code that the user code should call on timer expiry.</p> <p><u>Callback function prototype:</u> void timer_callback_t(void);</p> <p><u>Returns:</u> timer handle or NULL if timer start error</p>
stop_timer_t    *stop_timer	<p>Pointer function in user code used by HS NTP to stop a timer</p> <p><u>Prototype</u> void stop_timer_t(long timer_handle);</p> <p><u>Parameters:</u> Timer_handle – timer handle (returned from call to start_timer)</p>
event_callback_t *event_cb	<p>Pointer function in user code used by HS NTP to report operation results, status and errors. (Please see session 3)</p>

## 2.2 HsNtpDestroy

int HsNtpDestroy(void);

### DESCRIPTION

*This function de-initialises HS Ntp Library, releases all used resources and cleans up Socket Interface Layer.*

### PARAMETERS

Type	Name	Description
None	None	None

### RETURNS

- *HS\_NTP\_RC\_OK – success, HS NTP Library successfully de-initialised*
- *HS\_NTP\_RC\_NOTINIT – cannot destroy, HS NTP Library not yet initialised*

## 2.3 HsNtpGetErrStr

unsigned char \*HsNtpGetErrStr (int rc)

### DESCRIPTION

*This function is used to convert integer HS NTP return code into a readable string – error description*

### PARAMETERS

Type	Name	Description
Int	rc	HS NTP integer return code

### RETURNS

- *Pointer to zero terminated error string*
- *NULL – error not found (not a valid return code)*

## 2.4 HsNtpGetTime

```
int HsNtpGetTime(hs_ntp_session_t *s);
```

### DESCRIPTION

This function initiates NTP time request / response session with a remote NTP server. This function call is non-blocking. If the time request could be successfully sent, the function returns with code HS\_NTP\_RC\_OK. After HS NTP module receives and analyses time response from NTP server it shall call the event callback function in user code with an appropriate event and parameters to signal the completion of the operation.

\*s parameter is a pointer to session structure in user code which contains settings for this time request, such as server name or ip address and source UDP port number to use.

### PARAMETERS

Type	Name	Description
hs_ntp_session_t	*s	Pointer to session parameters structure:  srv_name – NTP server name (Note 1) srv_ip – NTP server IP address (Note 1) source_port - UDP source port to use  user_data – user data associated with this session (user session reference) not modified by NTP module  Note 1: Either server name or server IP address string can be supplied to the function. If srv_ip length is not 0, srv_ip string shall be used to send request to NTP server. If srv_ip length is 0, srv_name string shall be used to resolve server IP address first before connecting to it.

### RETURNS

- HS\_NTP\_RC\_NOTINIT – HS NTP Library not initialised
- HS\_NTP\_RC\_INV\_PAR – Invalid parameters specified
- HS\_NTP\_RC\_NONAME – NTP server name could not be resolved
- HS\_NTP\_RC\_TIMERFAIL – Timer start failure
- HS\_NTP\_RC\_BUSY – Time request in progress, new request cannot be started before it is complete
- HS\_NTP\_RC\_UDP\_SOCKET\_OPEN – Error opening UDP socket
- HS\_NTP\_RC\_UDP\_SOCKET\_SEND – Error sending UDP packet
- HS\_NTP\_RC\_OK - Success

### 3 HS NTP to USER Event Callback and Events

#### 3.1 Event Callback Prototype

```
typedef int event_callback_t(long handle, int ev, long arg1, long arg2);
```

Parameter	Description
user_handle	User handle, the same as specified in call to HsNtpGetTime in user_data parameter of hs_ntp_session_t structure
ev	Event code (see next section for list of event codes)
Arg1	Parameter 1, value depends on event
Arg2	Parameter 2, value depends on event

Returns:

True or False. For most events the return is insignificant and is not checked by HS NTP. Where HS NTP needs a specific return, it is clearly specified in this document

#### 3.2 Events

Event	Arg1	Arg2	Description
HS_NTP_USR_EV_SRVRESP	Pointer to structure hs_ntp_info_t. See details in section 3.2.1  Most significant information is offset in seconds required to apply to local clock in order to synchronize it to remote NTP server.	0	Time request completed successfully
HS_NTP_USR_EV_SRVRESPERR	HS_NTP_SRVERR_LEN – invalid NTP reply length  HS_NTP_SRVERR_UNSYNC – NTP server time is not synchronized to reliable source  HS_NTP_SRVERR_MODE – invalid mode	0	Error occurred processing NTP server response, see arg1 for detailed error code
HS_NTP_USR_EV_TIMEDOUT	0	0	Timed out waiting for server response.

### 3.2.1 NTP time reply structure (hs\_ntp\_info\_t)

Data Member	Description
T1	Time request sent by client (seconds, NTP timestamp) – informational purpose only
T2	Time request received at server (seconds, NTP timestamp) – informational purpose only
T3	Time reply sent by server (seconds, NTP timestamp) – informational purpose only
T4	Time reply received at client (seconds, NTP timestamp) – informational purpose only
offset_seconds	Calculated clock offset based on NTP reply in seconds. If application wishes to synchronize local system time to NTP server time, it should add Offset (in seconds) to local system time. Offset can take on both negative and positive values.
offset_milliseconds	Fraction part of seconds of the offset_seconds, converted to milliseconds. The application may adjust the local clock by this amount of milliseconds. This value is always positive. If offset_seconds is negative, the application can subtract offset_milliseconds from current system time, if the offset_seconds is positive, the application can add offset_milliseconds to current system time
st1	Value of T1 converted to SYSTEMTIME format
st2	Value of T2 converted to SYSTEMTIME format
st3	Value of T3 converted to SYSTEMTIME format
st4	Value of T4 converted to SYSTEMTIME format