Hillstone Software
Dublin 22, Republic of Ireland
www.hillstone-software.com
info@hillstone-software.com
Tel. +353 87 988 1568

HsTCPIPv4 Programming Manual

HsTCPIPv4 version 1.0

Document ver 1.0

26 August 2009

1 Introduction

HsTCPIPv4 is a suite ANSI C source code libraries which fully implement TCP IP protocol. HsTCPIPv4 as a whole or any of its included components can be used in an embedded system or on PC.

HsTCPIPv4 is supplied with full ANSI C source code and binaries.

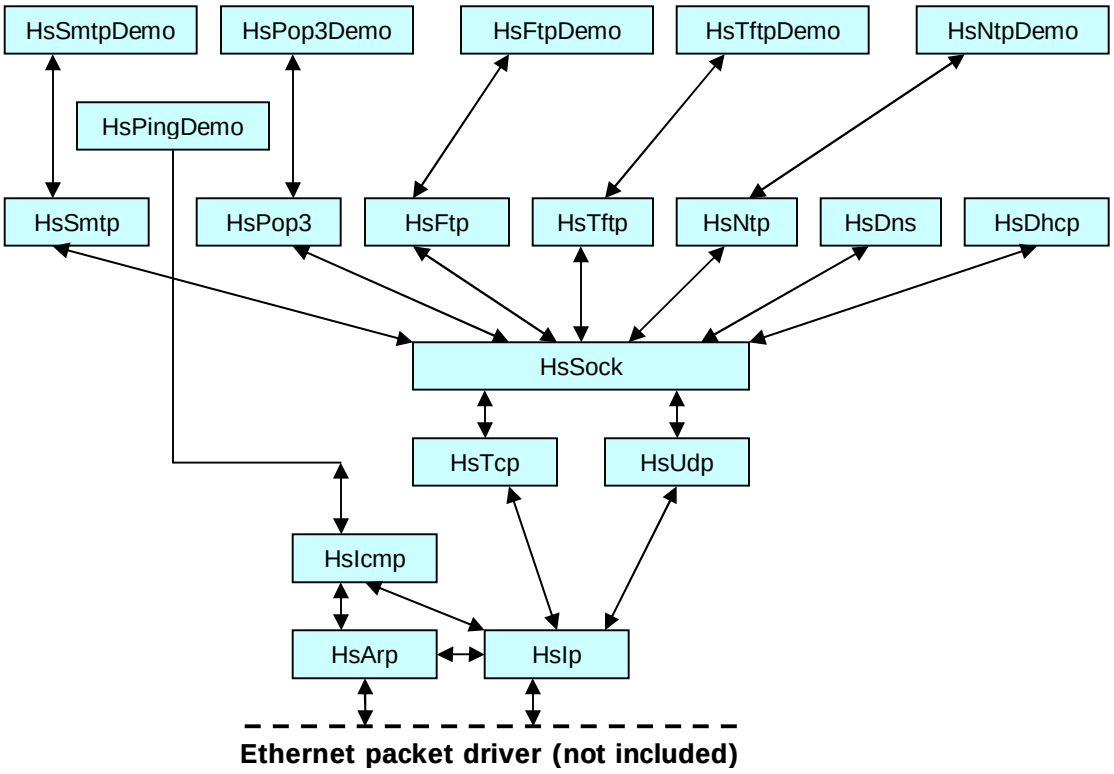
HsTCPIPv4 has been tested on both Little-endian (Intel x86) and Big-endian (Freescale / Motorola PowerQUICC**) memory architectures.**

HsTCPIPv4 includes the following components / protocols:

- **ARP**
- **ICMP**
- **IP**
- **UDP**
- **TCP**
- **DHCP / BOOTP**
- **DNS**
- **TFTP**
- **FTP**
- **NTP / SNTP**
- **POP3**
- **SMTP**
- **Sample applications:**
  ICMP Ping, NTP client,
  TFTP server and Client,
  FTP Client, POP3 Client,
  SMTP Client

2 HsTCPIPv4 API

2.1 Overall Software Architecture



HsTCPIPv4 is designed in a modular way, you can use only the specific modules you need for your application.

At the top level are sample applications supplied with HsTCPIPv4 protocol stack: SMTP client, POP3 client, FTP client, TFTP server and client, NTP client and ICMP Ping.

These applications use their respective library part of HsTCPIPv4: HsSmtp, HsPop3, HsFtp, HsTftp, HsNtp, HsIcmp

The above protocol modules interface to a common socket layer HsSock. HsSock library is talking to TCP (HsTcp) and UDP (HsUdp) modules

Both TCP and UDP modules and ICMP module interface to IP module (HsIp)
HsIp interfaces to ARP (HsArp) module. Both ARP and IP libraries interface to user supplied Ethernet packet driver.

HsTCPIPv4 protocol stack parameter configuration is done through HsSock module either using static parameters (IP address, Router IP address, subnet mask, DNS server IP address) or using dynamic configuration via HsDhcp library

All protocol modules use HsDns API if it is necessary to resolve target domain name to an IP address

2.2 Application Layer API

2.2.1 HsTftp

2.2.1.1 HsTftp Overview

HsTftp module implements Trivial File Transfer Protocol (TFTP) client and server side as per RFC 1350.

HS TFTP also implements RFC 1782 (TFTP Option Extension). The only supported TFTP option is block size as defined in RFC 1783 (TFTP Blocksize Option).

HS TFTP implements concurrent TFTP server and TFTP client operation.

2.2.1.2 HsTftp API

2.2.1.2.1 HsTftpInit

Declaration:
extern int HsTftpInit(tftp_init_t *init);

Summary:
This function initialises HS TFTP Library and must be called first before any other functions are called. Init structure contains function pointers which must be initialised with function addresses in application layer. HS TFTP module will call these functions when it needs to manage timers and memory.

Parameters:
hs_tftp_init_t *init - Pointer to initialisation structure, defined as follows:

parameter	Description
hs_tftp_get_buf_t *get_tx_buffer hs_tftp_get_buf_t *get_rx_buffer	**Prototype** unsigned char *hs_tftp_get_buf_t(long handle, unsigned int *length, int *cmd); **Parameters:** handle – application (user) layer context handle *length – Length of memory block. HS TFTP will pass number of bytes requested here. The user code sets the value to the actual number of bytes granted. While most of the time number of requested bytes equals number of given bytes, for example the last block may be shorter due to end of file, so the given length is less than requested. *cmd – this is intended for exchange of additional information or commands between user code and HS TFTP module – currently not used. . **Return:** pointer to memory buffer in user code or NULL if no memory available or nothing to give (end of transmission or end of file) **Description:** These functions shall be called from HS TFTP module when it needs to send next block of data (on reception of

	ACK from remote peer) or when data has been successfully received and now needs to be copied to user space from HS TFTP space.
hs_tftp_start_timer_t *start_timer_fn;	**Prototype:** long hs_tftp_start_timer_t(long handle, unsigned long secs); **Parameters:** handle – TFTP module context handle secs – number of seconds to timeout after. **Return:** Timer handle. Currently the same as application (user) context handle **Description:** This function in user code will be called from Hs TFTP code to start a timer with a specified number of seconds.
hs_tftp_stop_timer_t *stop_timer_fn	**Prototype:** void hs_tftp_stop_timer_t(long handle); **Parameters:** Handle – timer handle **Return:** No return **Description:** This function will be used by HS TFTP module to stop (destroy) a timer previously started with start_timer_fn.
unsigned short int max_blksize	**Maximum possible TFTP data block size, 0=use default, otherwise 8 to 65464 is the valid range.** This is a global setting which will affect all TFTP sessions. If TFTP blocksize option negotiation is used, HS TFTP will not negotiate or use blocksize above this value. This setting also affects the size of allocated buffer for receiving data from socket layer.

Return values:

Value	Description
HS_TFTP_RC_OK	Success
HS_TFTP_RC_INIT	HS TFTP already initialised
HS_TFTP_RC_INVALID_PAR	Invalid parameter
HS_TFTP_RC_NO_MEM	No free contexts or not enough memory for HS TFTP structures

Sample usage:

```
/*
 * initiase TFTP library
 */
int init_tftp_library(void)
{
        tftp_init_t init = {0};
        int     rc;

        init.get_tx_buffer          = tftp_get_buf_tx_cb;
```

```
        init.get_rx_buffer      = tftp_get_buf_rx_cb;
        init.start_timer_fn     = tftp_start_timer_cb;
        init.stop_timer_fn      = hs_tftp_stop_timer_cb;

        init.max_blksize= 65464;        // max blkoption option size value from RFC 2348

        rc = HsTftpInit(&init);

        if (rc == HS_TFTP_RC_OK)
                tftp_initialised = 1;

        return rc;
}
```

2.2.1.2.2 HsTftpDestroy

Declaration:

extern void HsTftpDestroy(void);

Summary:

De-allocates resources and closes HS TFTP services.

Parameters:

None

Return values:

None

Sample usage:

```
HsTftpDestroy();
```

2.2.1.2.3 HsTftpTransfer

Declaration:

```
extern
int HsTftpTransfer(int          operation,
        unsigned long   dest_ip,
        unsigned char   *filename,
        hs_tftp_ev_fn_t *callback_fn,
        unsigned short  tftp_port,

        unsigned short int blocksize,

        long            *handle,
        long            user_handle);
```

Summary:

Use this function to initiate client mode file transfer, either a send file transfer or receive file transfer. This function must be called after the library has been initialized with hsTftpInit

Parameters:

operation – integer operation code, one of the following:
TFTP_OP_SEND_FILE – Send file operation
TFTP_OP_GET_FILE – Receive file operation

dest_ip – destination IP address of TFTP server, (32 bit)

filename – pointer to null terminated string containing filename to send or receive

callback_fn – callback function to receive TFTP events related to this TFTP session or the following prototype:

typedef long hs_tftp_ev_fn_t(long handle, int ev_code, long arg1, long arg2);

handle passed to the above function is the same user_handle passed to HsTftpStartTransfer

ev_code TFTP event code, see table below.

tftp_port – destination server TFTP port (normally 69)

blocksize - 0=block size option not supported (default 512 bytes used)
otherwise blocksize to use in option negotiation, valid values 8 to 65464

handle - Connection handle returned after transfer initiated

user_handle – user context data to be stored in TFTP session context. This handle gets passed to event callback function (see above)

Return values:

Value	Description
HS_TFTP_RC_OK	Success
HS_TFTP_RC_NO_FREE_CTX	Maximum number of concurrent sessions reached, cannot allocate new session
HS_TFTP_RC_INVALID_PAR	Invalid parameter
HS_TFTP_RC_UDP_SOCK_OPEN	Error opening UDP socket
HS_TFTP_RC_UDP_SOCK_SEND	Error sending to UDP socket

TFTP session events:
The following table describes TFTP session events notified to the user via callback function callback_fn:

Event code	Description
HS_TFTP_EV_ERR_TMOUT	**TFTP session closed due to timeout** **Arg1 = 0** **Arg2 = 0**
HS_TFTP_RX_COMPLETE	**Receive file transfer complete** **Arg1 = 0** **Arg2 = 0**
HS_TFTP_EV_ERR_RXED	**TFTP error received from remote TFTP server, session**

	aborted. **Arg1 = integer error code as received from server** **Arg2 = pointer to null terminated string describing the error**
HS_TFTP_EV_ERR_OACK	**Session aborted. Invalid OACK (option ack) received**

Sample usage:

```
rc = HsTftpTransfer(TFTP_OP_GET_FILE, dest_ip,
      &current_filename[i], hs_tftp_ev_handler, ip_port, blksize, &pCtx->tftp_handle, (long)pCtx);

/*
 * Event calback from TFTP library
 */
long hs_tftp_ev_handler(long handle, int ev_code, long arg1, long arg2)
{
        unsigned char s[80] = {0};
        int len;
        unsigned char *errp = NULL;
        server_conn_ctx_t *pCtx;
        DWORD dwWritten;

        if (!tftp_initialised)
                return 0;

        pCtx = (server_conn_ctx_t *)handle;
        if (!pCtx) return 0;

        switch (ev_code)
        {
        case HS_TFTP_EV_ERR_TMOUT:
                write_event("Client session timeout");
                hs_tftp_cleanup_ctx(pCtx);
                break;

        case HS_TFTP_RX_COMPLETE:
                write_event("Client receive transfer complete");
                if (pCtx->rxbuf_len > 0)
                        WriteFile(pCtx->hFile, pCtx->rxbuf, pCtx->rxbuf_len, &dwWritten, NULL);
                hs_tftp_cleanup_ctx(pCtx);
                break;

        case HS_TFTP_EV_ERR_RXED:
                errp = (unsigned char *)arg2;

                len = strlen(errp);
                if (len > ((sizeof s)-1))
                {
                        len = ((sizeof s)-1);
                }
                memcpy(s, errp, len);

                write_event(s);
```

```
                hs_tftp_cleanup_ctx(pCtx);
                break;

        case HS_TFTP_EV_ERR_OACK:
                write_event("Client session closed: invalid OACK rxed");
                hs_tftp_cleanup_ctx(pCtx);
                break;
        }

        return 0;
}
```

2.2.1.2.4 HsTftpAbort

Declaration:
extern int HsTftpAbort(long handle);

Summary:
Abort current operation and cleanly disconnect remote end based on passed connection handle. This function is going to send TFTP ERROR packet to remote end with the string "Aborted by user" and cleanup local TFTP session context.

Parameters:
handle – TFTP session handle

Return values:

Value	Description
HS_TFTP_RC_OK	Success
HS_TFTP_RC_NOT_INIT	HSTFTP is library not initialized
HS_TFTP_RC_INVALID_PAR	Invalid parameter (null handle)

Sample usage:

```
HsTftpAbort(pCtx->tftp_handle);
```

2.2.1.2.5 HsTftpTimerExpired

Declaration:
extern void HsTftpTimerExpired(long timer_handle);

Summary:
Function called from user code when timer previously started by HS TFTP has expired

Parameters:
timer_handle - timer handle

Return values:
None

Sample usage:

```
/*
```

```
 * windows timer callback
 */
TIMERPROC TimerProc(HWND hwnd,  UINT uMsg, UINT_PTR idEvent,  DWORD
dwTime)
{
        KillTimer(hwnd, idEvent);

        HsTftpTimerExpired((long)idEvent);
        return 0;
}
```

2.2.1.2.6 HsTftpStartServer

Declaration:

```
extern int HsTftpStartServer(
        hs_tftp_srv_ev_fn_t *callback_fn,
        unsigned short int blocksize,
        unsigned short tftp_port);
```

Summary:

Starts server operation of HS TFTP module.

Parameters:

unsigned short int blocksize – controls the handling of blocksize option received in WRQ and RRQ requests (file read and write requests) from clients. Set to 0 to not support the blocksize option and always use default block size of 512 bytes. Set to maximum supported block size in the range from 8 to 65464. Please note that this value also cannot exceed the value of max_blksize set in init structure when HsTftpInit was called. If client request contains TFTP blocksize option, the request will be acknowledged by HS TFTP server with OACK packet containing either the requested blocksize value or a lower value if blocksize parameter in this call is lower.

unsigned short tftp_port – UDP port number the HS TFTP module listens for incoming TFTP client commands, recommended default value is 69.

hs_tftp_srv_ev_fn_t *callback_fn – function pointer to callback function in application layer which receives events related to the status of the server operations:

typedef long hs_tftp_srv_ev_fn_t(long handle, int ev_code, long arg1, long arg2);

handle – currently unused
ev_code – one of the following values:

HS_TFTP_EV_WRITE_REQ – write request received from remote TFTP client (client wants to send file)
Arg1 – pointer to null terminated filename string
Arg2 – 32 bit remote IP address (address of TFTP client sending this request)

HS_TFTP_EV_READ_REQ - read request received from remote TFTP client (client wants to get file)
Arg1 – pointer to null terminated filename string
Arg2 – 32 bit remote IP address (address of TFTP client sending this request)

Return values:

Value	Description
HS_TFTP_RC_OK	Success
HS_TFTP_RC_UDP_SOCK_OPEN	UDP layer failed to open session on specified UDP port
HS_TFTP_RC_INVALID_PAR	Invalid parameter

Sample usage:

```
/*
 * Event calback from TFTP library (server mode)
 */
long hs_tftp_ev_server_handler(long handle, int ev_code, long arg1, long arg2)
{
        unsigned char *file;

        if (!tftp_initialised)
                return 0;

        if (!handle)
                return 0;

        switch (ev_code)
        {
        case HS_TFTP_EV_WRITE_REQ:
                file = (unsigned char *)arg1;
                process_file_write_request(file, arg2);
                break;


        case HS_TFTP_EV_READ_REQ:
                file = (unsigned char *)arg1;
                process_file_read_request(file, arg2);
                break;
        }

        return 0;
}

/* Start server mode */
void StartServer(void)
{
        int rc;

        if (!tftp_initialised)
        {
                rc = init_tftp_library();
                if (rc != HS_TFTP_RC_OK)
                {
                        printf("HsTFTP init failed. Error %d\n", rc);
                        return;
                }
        }
```

```
        if (!server_started)
        {
                rc = HsTftpStartServer(hs_tftp_ev_server_handler, 65464, TFTP_PORT);
                if (rc != HS_TFTP_RC_OK)
                {
                        printf("Server failed to start. HS TFTP Error: %d\n", rc);
                        return;
                }
                server_started = TRUE;
        }
        else
        {
                printf("Server mode already running\n");
                return;
        }

        testing_hs_tftp = 1;

        printf("Server mode started OK\n");
}
```

2.2.1.2.7 HsTftpServerStartReceive

Declaration:

```
extern
int HsTftpServerStartReceive(
        hs_tftp_ev_fn_t   *callback_fn,    // event callback (used for indication of completion or error)
        long              *handle,         // Connection handle returned after transfer initiated
        long              user_handle);    // upper layer context handle
```

Summary:

Start receiving requested file from remote peer (in Server mode). This function is called in response to HS_TFTP_EV_WRITE_REQ event, which occurs when a write request for a file is received from remote TFTP client

Parameters:

hs_tftp_ev_fn_t *callback_fn – function pointer to a callback function in application (user) layer code which receives notifications related to this transfer session.

typedef long hs_tftp_ev_fn_t(long handle, int ev_code, long arg1, long arg2);
handle – application layer handle
ev_code – event, one of the following:
HS_TFTP_EV_ERR_TMOUT – session timed out and closed
HS_TFTP_RX_COMPLETE – receive transfer completed successfully
Arg1 and arg2 are currently unused

long *handle – pointer to long variable to received TFTP module connection handle after file transfer is initiated.

long user_handle – application layer context handle. This handle is saved into corresponding TFTP session context and is returned unchanged as a parameter in various notification callbacks from HS TFTP to application layer code.

Return values:

Value	Description
HS_TFTP_RC_OK	**Success**
HS_TFTP_RC_UDP_SOCK_OPEN	**UDP layer failed to open session**
HS_TFTP_RC_INVALID_PAR	**Invalid parameter**

Sample usage:

```
/* server mode - process file write request */
void process_file_write_request(unsigned char *filename,  long arg2)
{
        int                             rc;
        unsigned char      ipstr[20] = {0};
        server_conn_ctx_t *pCtx;

        HsSockInetNtoa(arg2, ipstr);
        printf("file write request from (%s) %s\n", ipstr, filename);

        pCtx = tftp_alloc_srvconn_ctx();
        if (!pCtx)
        {
                printf("rejected: no free contexts\n");
                HsTftpRejectRq(HS_TFTP_EV_WRITE_REQ,
                                HS_TFTP_SERVER_ERR_USER, "no free contexts");
                return;
        }

        pCtx->hFile = CreateFile(filename,          // file to open
            GENERIC_WRITE,                          // open for writing
            0,                                              // no sharing
            NULL,                                   // default security
            OPEN_ALWAYS,                            // overwrite existing file
            FILE_ATTRIBUTE_NORMAL,                  // normal file
            NULL);

        if (pCtx->hFile == INVALID_HANDLE_VALUE)
                pCtx->hFile = NULL;

        if (!pCtx->hFile)
        {
                printf("rejected: file open error\n");

                tftp_free_srvconn_ctx(pCtx);

                /* Reject request */
                HsTftpRejectRq(HS_TFTP_EV_WRITE_REQ, HS_TFTP_SERVER_ERR_FIO, NULL);
                return;
        }

        pCtx->is_server_session = TRUE;
        pCtx->is_send = FALSE;

        tftp_add_srvconn(pCtx);

        rc = HsTftpServerStartReceive(hs_tftp_server_ev_handler, &pCtx->tftp_handle, (long)pCtx);
        if (rc != HS_TFTP_RC_OK)
        {
                printf("Server Start Receive failed RC (%u)\n", rc);
```

```
                hs_tftp_cleanup_ctx(pCtx);
                return;
        }

        printf("Server receive transfer started\n");
}
```

2.2.1.2.8 HsTftpServerStartSend

Declaration:

```
extern int HsTftpServerStartSend(
        hs_tftp_ev_fn_t  *callback_fn,    // event callback (used for infication of completion or error)
        long             *handle,         // Connection handle returned after transfer initiated
        long             user_handle);    // upper layer context handle
```

Summary:

Start sending requested file to remote peer (in Server mode). This function is called in response to HS_TFTP_EV_READ_REQ event, which occurs when a read request for a file is received from remote TFTP client

Parameters:

hs_tftp_ev_fn_t *callback_fn – function pointer to a callback function in application (user) layer code which receives notifications related to this transfer session.

typedef long hs_tftp_ev_fn_t(long handle, int ev_code, long arg1, long arg2);
handle – application layer handle
ev_code – event, one of the following:
HS_TFTP_EV_ERR_TMOUT – session timed out and closed
HS_TFTP_RX_COMPLETE – receive transfer completed successfully
Arg1 and arg2 are currently unused

long *handle – pointer to long variable to received TFTP module connection handle after file transfer is initiated.

long user_handle – application layer context handle. This handle is saved into corresponding TFTP session context and is returned unchanged as a parameter in various notification callbacks from HS TFTP to application layer code.

Return values:

Value	Description
HS_TFTP_RC_OK	Success
HS_TFTP_RC_UDP_SOCK_OPEN	UDP layer failed to open session
HS_TFTP_RC_INVALID_PAR	Invalid parameter

Sample usage:

```
/* server mode - process file read request */
void process_file_read_request(unsigned char *filename, long arg2)
{
        int rc;
        long fsize = 0;
        unsigned char ipstr[20] = {0};
```

```
        server_conn_ctx_t *pCtx;

        HsSockInetNtoa(arg2, ipstr);
        printf("file read request from (%s) %s\n", ipstr, filename);

        pCtx = tftp_alloc_srvconn_ctx();
        if (!pCtx)
        {
                printf("rejected: no free contexts\n");
                HsTftpRejectRq(HS_TFTP_EV_READ_REQ,
                        HS_TFTP_SERVER_ERR_USER, "no free contexts");
                return;
        }

        pCtx->hFile = CreateFile(filename, GENERIC_READ, FILE_SHARE_READ, NULL,
                                                        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
        if (pCtx->hFile == INVALID_HANDLE_VALUE)
                pCtx->hFile = NULL;

        fsize = (long)GetCompressedFileSize(filename, NULL);
        if ((!pCtx->hFile) || (!fsize))
        {
                printf("rejected: file open error\n");
                HsTftpRejectRq(HS_TFTP_EV_READ_REQ, HS_TFTP_SERVER_ERR_FNOTFOUND, NULL);
                hs_tftp_cleanup_ctx(pCtx);
                return;
        }

        pCtx->fblock = pCtx->rxbuf;

        pCtx->is_server_session = TRUE;
        pCtx->is_send = TRUE;

        tftp_add_srvconn(pCtx);

        rc = HsTftpServerStartSend(hs_tftp_server_ev_handler, &pCtx->tftp_handle, (long)pCtx);
        if (rc != HS_TFTP_RC_OK)
        {
                printf("Server Start Send failed RC (%u)\n", rc);
                hs_tftp_cleanup_ctx(pCtx);
                return;
        }

        pCtx->total_blocks = fsize / TFTP_BLK_SIZE;

        printf("Server send transfer started\n");
}
```

2.2.1.2.9 HsTftpErrStr

Declaration:

extern unsigned char *HsTftpErrStr(int rc);

Summary:

Returns a pointer to null terminated error string which describes TFTP error code rc passed to the function.

Parameters:

rc – error code returned by one of HSTFTP functions

Return values:

Returns a pointer to null terminated error string which describes TFTP error code rc passed to the function.

Sample usage:

```
rc = HsTftpTransfer(TFTP_OP_SEND_FILE, dest_ip,
                                    &current_filename[i], hs_tftp_ev_handler, ip_port, blksize, &pCtx->tftp_handle, (long)pCtx);

if (rc != HS_TFTP_RC_OK)
{
        memset(s, 0, sizeof s);
        sprintf(s,"Client: cannot send file: HS TFTP Error: %s", HsTftpErrStr(rc));
        write_event(s);
        hs_tftp_cleanup_ctx(pCtx);
        return;
}

write_event("Client send transfer started");
```

2.2.1.2.10 HsTftpRejectRq

Declaration:
extern void HsTftpRejectRq(int rq, int reason, unsigned char *str);

Summary:
Sends TFTP ERROR packet (in server mode) to remote TFTP client with specified reason code and descriptive ASCII string. This function may be called in response to HS_TFTP_EV_WRITE_REQ and HS_TFTP_EV_READ_REQ events.

Parameters:
rq – currently unused

reason – reason code, one of the following:
HS_TFTP_SERVER_ERR_FIO - FILE I/o error
HS_TFTP_SERVER_ERR_FNOTFOUND - file not found
HS_TFTP_SERVER_ERR_USER - user defined error, send supplied error string

Str – pointer to zero terminated ASCII string to send with the ERROR packet, only valid if reason is HS_TFTP_SERVER_ERR_USER.

Return values:
none

Sample usage:

```
HsTftpRejectRq(HS_TFTP_EV_WRITE_REQ, HS_TFTP_SERVER_ERR_USER, "no free contexts");
```

2.2.1.3 HsTftp Application Notes

2.2.1.3.1 Model of Operation

When user application initialises Hs TFTP library, it provides interface callbacks for the services used by HS TFTP protocol module: timer management, memory management and event callbacks. This architecture makes it easy to port HS TFTP protocol module to any environment. HS TFTP internally at a lower layer interfaces to HS Sock library which provides UDP transport services. User application need not worry about Winsock - HS TFTP does all transmission, reception and event handling over socket layer.

TFTP module handles all protocol information flow, error recovery, acknowledgements, timeouts and so on. When it is appropriate to send next block of data HS TFTP will get next memory block from user application. Similarly, when data has been received HS TFTP module will get the next block of memory from user application to store data into.

2.2.1.3.2 Sending File Considerations

Client mode: To send file in client mode, setup all callbacks and call HsTftpTransfer. Every time, HS TFTP receives an acknowledgement from remote end, it will call get_tx_buffer callback function – give it the next data block to transmit. After reception of acknowledge from remote end for last data block, HS TFTP is going to call the get_tx_buffer callback again – give it NULL pointer and zero length and consider transfer complete.

Server mode: The procedure for sending files in server mode in similar to client mode, except it is initiated not with HsTftpStart transfer, but with HsTftpServerStartSend

2.2.1.3.3 Receiving File Considerations

Client mode: To receive file in client mode, setup all callbacks and call HsTftpTransfer. Every time HS TFTP receives, correctly processes and acknowledges a packet from remote end, it will call get_rx_buffer callback function – give it the pointer of the next space to store the received data into. When HS TFTP receives the last data packet from remote end it will call event callback function with HS_TFTP_RX_COMPLETE return code. At this point consider receive transfer complete.

Server mode: The procedure for receiving files in server mode in similar to client mode, except it is initiated not with HsTftpStart transfer, but with HsTftpServerStartReceive

2.2.2 HsFtp

2.2.2.1 HsFtp Overview

HsFtp library implements the client side of the File Transfer Protocol over TCP socket layer according to RFC 959.

The library allows a user application to connect to remote FTP servers, traverse server directory structure, send and receive files

HS FTP Client Library incorporates the necessary server response processing and state machine required to comply with a simple implementation of FTP client.

The following FTP command sequences are supported:

- "USER" - authentication
- "PASS" - authentication
- "PASV" – establish passive mode data connection
- "ABOR" - abort
- "LIST" – request listing
- "CWD" – change directory
- "MKD" – create directory
- "RMD" – remove directory
- "TYPE" – set transfer type
- “RETR" – receive file
- "STOR" – transmit file
- "DELE" – delete file
- "NOOP" – no operation
- "PWD" – request current directory name
- "RNFR" – rename from
- "RNTO" – rename to

Additionally, HS FTP source code package contains “recursive folder operations” module (HsFtpRecurs) which implements:

- recursive folder download (download folder with all files and sub-folders)
- recursive folder upload (upload folder with all files and sub-folders)
- recursive folder delete (delete folder with all files and sub-folders)

2.2.2.2 HsFtp API

2.2.2.2.1 HsFtpInit

Declaration:
int HsFtpInit(hs_ftp_init_t *pInit)

Summary:
This function initialises HS FTP Library and MUST be called once, prior to calling any other library functions

Parameters:

hs_ftp_init_t *pInit – pointer to initialization structure of type hs_ftp_init_t. This structure is described as follows:

Parameter name	Description
***start_timer**	**Pointer to start timer callback function. HSFTP calls this function whenever it needs to start a timer** **Start Timer callback function is defined as:** typedef long hsftp_timer_start_t(unsigned long timeout_ms, void *arg, hsftp_timer_cb_t *cb); **timeout_ms – timeout value in milliseconds** **arg – void parameter argument, must be passed back in callback function cb** **cb – callback function to be called when timer expires: defined as typedef void hsftp_timer_cb_t(void *arg);**
***stop_timer**	**Pointer to stop timer callback function. HSFTP calls this function whenever it needs to stop a previously started timer.** **Stop timer function has the following prototype:** **typedef void hsftp_timer_stop_t(long timer_han);** **timer_han – timer handle**
create_file	**Pointer to create file callback function. HSFTP calls this function when it needs to open a disk file for reading or for writing.** **It has the following prototype:** void *hsftp_create_file_t(unsigned char *filename, int mode, int open_always); **filename – name of file to open** **mode – access mode:** **HSFTP_FILE_READ – read access** **HSFTP_FILE_WRITE – write access** **open_always: 1 = Open file always regardless of file existed before or not; 0=normal operation** **returns file handle cast to void pointer.**
close_file	**Pointer to function callback to close a file. HSFTP calls this function when it needs to close a file.** **It has the following prototype:** **void hsftp_close_file_t(void *hFile);** **hFile – file handle (obtained with create_file call)**

write_file	**Pointer to function callback to write file data. HSFTP calls this function whenever it needs to save next block of data to an open file.** **It has the following prototype:** int hsftp_write_file_t(void *hFile, void *pData, int length, int *bytes_written); hFile – file handle pData – pointer to data buffer to write int length – length of data buffer in bytes to write bytes_written – pointer to integer to receive actual number of bytes written to file returns 1: operation successful; 0: file write error occurred
read_file	**Pointer to function callback to read file data. HSFTP calls this function whenever it needs to read next block of data from an open file.** **It has the following prototype:** int hsftp_read_file_t(void *hFile, void *pData, int length, int *bytes_read); hFile – file handle pData – pointer to data buffer to read into int length – length of data buffer in bytes to read bytes_read – pointer to integer to receive actual number of bytes read returns 1: operation successful; 0: file write error occurred
getticks	**Pointer to function callback to obtain current number of millisecond ticks since system boot-up.** **It has the following prototype:** **typedef unsigned long hs_ftp_get_ticks_fn_t(void);** **returns number of milliseconds since bootup**

Return values:

Value	Description
HS_FTP_RC_OK	**Success**
HS_FTPCLI_RC_INV_PAR	**Invalid parameter**
HS_FTPCLI_RC_ALRINIT	**HsFtp is already initialized**

Sample usage:

```
hs_ftp_init_t sInit = {0};

        sInit.start_timer = sock_timer_start;
        sInit.stop_timer = sock_timer_stop;
        sInit.create_file = hstfp_create_file;
        sInit.close_file = hstfp_close_file;
        sInit.read_file = hstfp_read_file;
        sInit.write_file = hstfp_write_file;
        sInit.getticks = hs_sock_get_ticks;

    HsFtpInit(&sInit);


static long sock_timer_start(unsigned long timeout_ms, void *arg, hs_sock_timer_cb_t *cb)
{
        timerbock_t *pTmrBlk;

        pTmrBlk = (timerbock_t *)alloc_timer();
        if (!pTmrBlk) return 0;

        pTmrBlk->arg = arg;
        pTmrBlk->cb = cb;
        pTmrBlk->ms_count = 0;
        pTmrBlk->ms_timeout = timeout_ms;

        pTmrBlk->enabled = TRUE;

        return (long)pTmrBlk;
}

static void sock_timer_stop(long timer_han)
{
        timerbock_t *pTmrBlk;

        pTmrBlk = (timerbock_t *) timer_han;

        if (!pTmrBlk) return;

        if (!pTmrBlk->allocated) return;

        if (!pTmrBlk->enabled) return;

        pTmrBlk->allocated = FALSE;
        pTmrBlk->enabled = FALSE;
}

/*
 * Create file callback
 */
static void *hstfp_create_file(unsigned char *filename, int mode, int open_always)
{
        HANDLE hFile;
        DWORD accessmode;

        switch (mode)
```

```
	{
	case HSFTP_FILE_READ:
		accessmode = GENERIC_READ;
		break;

	case HSFTP_FILE_WRITE:
		accessmode = GENERIC_WRITE;
		break;

	default:
		return NULL;
	}

	hFile = CreateFile(filename,		// file to open
	   accessmode,				// open for writing
	   0,					// no sharing
	   NULL,				// default security
	   (open_always) ? OPEN_ALWAYS : OPEN_EXISTING,
	   FILE_ATTRIBUTE_NORMAL,		// normal file
	   NULL);

	if (hFile == INVALID_HANDLE_VALUE)
		return NULL;

	return (void *)hFile;
}

/*
 * Close file callback
 */
void hstfp_close_file(void *hFile)
{
	CloseHandle((HANDLE)hFile);
}

/*
 * Read file callback
 */
int hstfp_read_file(void *hFile, void *pData, int length, int *bytes_read)
{
	DWORD dwBytesRead = 0;
	BOOL bResult;

	bResult = ReadFile((HANDLE)hFile, pData, (DWORD)length, &dwBytesRead, NULL);

	*bytes_read = (int)dwBytesRead;

	return (int)bResult;
}

/*
 * Write file callback
 */
int hstfp_write_file(void *hFile, void *pData, int length, int *bytes_written)
{
	DWORD dwBytesWritten = 0;
```

```
	BOOL bResult;

	bResult = WriteFile((HANDLE)hFile, pData, (DWORD)length, &dwBytesWritten, NULL);

	*bytes_written = (int)dwBytesWritten;

	return (int)bResult;
}

static unsigned long hs_sock_get_ticks(void)
{
	return (unsigned long)GetTickCount();
}
```

2.2.2.2.2 HsFtpCleanUp

Declaration:
int HsFtpCleanUp (void)

Summary:
This function de-initialises HS FTP Library and releases resources used by it. Any active control and data connections shall be disconnected and all contexts and any used memory freed

Parameters:
none

Return values:

Value	Description
HS_FTP_RC_OK	**Success**
HS_FTPCLI_RC_NOTINIT	library not initialised

Sample usage:

```
HsFtpCleanUp();
```

2.2.2.2.3 HsFtpTick

Declaration:
int HsFtpTick(void)

Summary:
This function must be called as often as possible from the main program loop. This function drives internal timers and socket layer operations used by HS FTP module

Parameters:
none

Return values:

Value	Description
HS_FTP_RC_OK	**Success**
HS_FTPCLI_RC_NOTINIT	library not initialised

Sample usage:

```
while (1)
{
	if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
	{
		if (GetMessage(&msg, NULL, 0, 0))
		{
			if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
			{
				TranslateMessage(&msg);
				DispatchMessage(&msg);
			}
		}
		else
		{
			break;
		}
	}
	else
	{
		HsFtpTick();
		HsFtpRecursTick();

		Sleep(1);

		update_stats();
	}
}
```

2.2.2.2.4 HsFtpCliConnect

Declaration:
extern int HsFtpCliConnect(hs_ftp_conn_t *conn, long *session_handle)

Summary:
This function is used to connect to remote FTP server

Parameters:
hs_ftp_conn_t *conn - Pointer to structure in user code which contains connection parameters. Please see the next section below for detailed description of data members.

Description of hs_ftp_conn_t structure

Data Member	Description
unsigned char *srv_name;	Remote FTP server DNS name to connect to, for example (ftp.hillstone-software.com). This can also be an IP address string in dotted format, for example “192.168.1.2”. This parameter is a pointer to null terminated string.
unsigned short srv_port;	Remote FTP server port to connect to
unsigned char *usrname;	FTP account user name for authentication, pointer to null

	terminated string
unsigned char *password;	FTP account password for authentication, pointer to null terminated string
ftp_callback_t *callback;	Pointer to callback function used by HS FTP to communicate to user application. Please see the detailed description in section 3 (HS FTP Client Module to USER Event Callback and Events)
void *user_ref;	User data. The user application can store any value (or pointer) here. HS FTP module shall always pass this value back unmodified to user event callback function

long *session_handle - Pointer to long variable In user code to receive HS FTP module handle associated with this FTP session. This handle must then be used in all further calls to HS FTP module related to this FTP session.

Return values:

Value	Description
HS_FTP_RC_OK	Success - HS FTP module started session establishment to remote FTP server. The actual result shall be asynchronously indicated via user callback event, part of hs_ftp_conn_t structure, see description in the following section
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NO_CTX	there are no more free HS FTP Client contexts, maximum number of concurrent connections reached
HS_FTPCLI_RC_TCPCONNFAIL	outgoing TCP connection to remote FTP server failed

Sample usage:

```
memset(&conn, 0, sizeof(hs_ftp_conn_t));

memset(server_name, 0, sizeof server_name);
GetDlgItemText(hDlg, IDC_EDIT_NAME, server_name, (sizeof (server_name) -1));

conn.srv_name = server_name;

conn.srv_port = (unsigned short)GetDlgItemInt(hDlg, IDC_EDIT_PORT, &translated, FALSE);
if (!translated)
{
        PutLog(hDlg, "Cannot connect: no port specified");
        return;
}

memset(username, 0, sizeof (username));
memset(password, 0, sizeof (password));

GetDlgItemText(hDlg, IDC_EDIT_USER, username, sizeof(username) - 1);
GetDlgItemText(hDlg, IDC_EDIT_PW, password, sizeof(password) - 1);

conn.usrname = username;
conn.password = password;

conn.callback = hs_ftp_callback;
conn.user_ref = (void *)&myuserref;
```

```
rc = HsFtpCliConnect(&conn, &ftp_session);
if (rc != HS_FTP_RC_OK)
{
    state = APP_STATE_IDLE;
    sprintf(s, "HS FTP error: %s", HsFtpGetErrStr(rc));
    write_status(hDlg, RED, s);
    PutLog(hDlg, s);
    return;
}

state = APP_STATE_CONNECTING;
write_status(hDlg, GREEN, "Connecting");
PutLog(hDlg, "Connecting");
```

2.2.2.2.5 HsFtpCliDisconnect

Declaration:
extern int HsFtpCliDisconnect(long session_handle)

Summary:
This function is used to close FTP session to remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated

Sample usage:

```
HsFtpCliDisconnect(ftp_session);
```

2.2.2.2.6 HsFtpCliChDir

Declaration:
extern int HsFtpCliChDir(long session_handle, unsigned char *dir);

Summary:
This function is used to change to a different directory on remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *dir - Directory name to change to, null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to change to a new directory on remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

```
rc = HsFtpCliChDir(ftp_session, s);
```

2.2.2.2.7 HsFtpCliCreateDir

Declaration:
extern int HsFtpCliCreateDir(long session_handle, unsigned char *dir);

Summary:
This function is used to create a new directory on remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *dir - Directory name to change to, null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to create to a new directory on remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

```
rc = HsFtpCliCreateDir(ftp_session, editstr);
```

2.2.2.2.8 HsFtpCliRemoveDir

Declaration:
extern int HsFtpCliRemoveDir(long session_handle, unsigned char *dir);

Summary:
This function is used to remove a directory on remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *dir - Directory name to change to, null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to create to remove a directory on remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliRemoveDir(ftp_session, editstr);

2.2.2.2.9 HsFtpCliList

Declaration:
extern int HsFtpCliList(long session_handle);

Summary:
This function is used to receive directory file listing from the remote FTP server. The listing is returned asynchronously via user event callback – event HS_FTPCLI_USR_EV_LIST, , please see the detailed description in section (HS FTP Client Module to USER Event Callback and Events). The listing is returned in the same format as it came in from the server, the actual format depends on server OS and FTP software. Please see the example of parsing the listing (breaking down into filenames) supplied in HS FTP Demo application, function load_remote_directory.

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to receive directory listing from remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event

	Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated

Sample usage:

rc = HsFtpCliList(ftp_session);

2.2.2.2.10 HsFtpCliGetFile

Declaration:
extern int HsFtpCliGetFile(long session_handle, unsigned char *filename, __int64 fsize)

Summary:
This function is used to transfer the specified file from remote FTP server to local system.

Successful return of this function indicates that HS FTP has started file reception protocol sequence.

HS FTP handles all aspects of receiving the file (opening disk file, receiving and writing blocks of data and closing file). As the file data gets transferred block by block, events HS_FTPCLI_USR_EV_RX_STATUS are generated, informing the application that the next block of data (specifying its size) has been written from to file.

Successful completion of file transfer (when file has been completely received and disk file closed) is reported to application via HS_FTPCLI_USR_EV_RXDONE event

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *filename - Filename to get, pointer to null terminated string

__int64 fsize - Size of file to get, 64 bit integer. File size shall be obtained first after successful call to HsFtpCliList

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to start downloading the specified file from remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliGetFile(ftp_session, rxfname, remote_fsz);

2.2.2.2.11 HsFtpCliSendFile

Declaration:
extern int HsFtpCliSendFile (long session_handle, unsigned char *filename)

Summary:
This function is used to transfer the specified file from local system to remote FTP server.

Successful return of this function indicates that HS FTP has started file sending protocol sequence.

HS FTP handles all aspects of sending the file (opening disk file, reading blocks of data and sending them, closing file). As the file data gets transferred block by block, events HS_FTPCLI_USR_EV_TX_STATUS are generated, informing the application that the next block of data (specifying its size) has been read from disk file and sent to FTP server

Successful completion of file transfer (when file has been completely sent and disk file closed) is reported to application via HS_FTPCLI_USR_EV_TXDONE event

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *filename - Filename to get, pointer to null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to start uploading the specified file to remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliSendFile(ftp_session, txfname);

2.2.2.2.12 HsFtpCliDeleteFile

Declaration:
extern int HsFtpCliDeleteFile(long session_handle, unsigned char *filename);

Summary:
This function is used to delete a file on remote FTP server

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char - *filename - Filename to get, pointer to null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to delete the specified file from remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliDeleteFile(ftp_session, editstr);

2.2.2.2.13 HsFtpCliAbort

Declaration:
extern int HsFtpCliAbort (long session_handle)

Summary:
This function is used to abort current FTP operation in progress

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to abort current command. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated

Sample usage:

rc = HsFtpCliAbort(ftp_session);

2.2.2.2.14 HsFtpCliRename

Declaration:
extern int HsFtpCliRename(long session_handle, unsigned char *oldname, unsigned char *newname)

Summary:
This function is used to rename a file or folder on remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char * oldname - File or folder name to rename, null terminated string

unsigned char *newname - New name for file or folder name, null terminated string

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to rename a file or folder *on* remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliRename(ftp_session, oldpath, editstr);

2.2.2.2.15 HsFtpCliGetCurrentDirectory

Declaration:
extern int HsFtpCliGetCurrentDirectory(long session_handle);

Summary:
This function is used to request the current working directory name from remote FTP server

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated

	procedure to request the current working directory from remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliGetCurrentDirectory(ftp_session);

2.2.2.2.16 HsFtpCliNoop

Declaration:

extern int HsFtpCliNoop(long session_handle);

Summary:
This function is used to send NOOP command to remote FTP server. This command does not require the server to execute any action except send a valid response

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

Return values:

Value	Description
HS_FTP_RC_OK	Success, session disconnected, HS FTP initiated procedure to send NOOP command to remote FTP server. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in (HS FTP Client Module to USER Event Callback and Events)
HS_FTPCLI_RC_NOTINIT	library not initialised
HS_FTPCLI_RC_INV_PAR	invalid parameters supplied
HS_FTPCLI_RC_NOTALLOC	invalid session – not allocated
HS_FTPCLI_RC_INVALID_CMD	command is invalid for current session state

Sample usage:

rc = HsFtpCliNoop(ftp_session);

2.2.2.2.17 HsFtpSetConfig

Declaration:
extern void HsFtpSetConfig (hs_ftp_config_t *cfg);

Summary:
This function is used to set HS FTP global configuration parameters

Parameters:
hs_ftp_config_t *cfg - Pointer to configuration structure hs_ftp_config_t, with the following data members: ong t1_timeout – timeout in milliseconds for all HS FTP operations. Default timeout is 15000 milliseconds

Return values:

None

Sample usage:

```
T1 = val;
cfg.t1_timeout = (long)T1;
HsFtpSetConfig(&cfg);
```

2.2.2.2.18 HsFtpGetStats

Declaration:
extern void HsFtpGetStats (hsftp_stats_t *pStats);

Summary:
This function is used to set HS FTP global configuration parameters

Parameters:
hsftp_stats_t *pStats - Pointer to stats structure with the following data members:

```
__int64    total_bytes_Tx;      // total number of bytes transmitted
__int64    total_bytes_Rx;      // total number of bytes received
```

These counters include both control and data FTP connections

Return values:

None

Sample usage:

```
hsftp_stats_t stats = {0};

HsFtpGetStats(&stats);
```

2.2.2.3 HS FTP Client Module to User Event Callback and Events

2.2.2.3.1 Event Callback Prototype

typedef int ftp_callback_t(void *user_ref, int ev, long arg1, long arg2, long arg3);

Parameter	Description
void *user_ref	User data. The user application passes any value (or pointer) in the call to HsFtpCliConnect. HS FTP module always pass this value back unmodified to this parameter, which can be used by user code to identify FTP session contexts.
int ev	Event id, the full list and description is provided in section 3.2 Events.
long arg1	Parameter 1, specific to event id
long arg2	Parameter 2, specific to event id
long arg3	Parameter 3, specific to event id

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

2.2.2.4 Events

Event	Description
HS_FTPCLI_USR_EV_LOGGEDIN	FTP session to remote FTP server established, login successful. The user application can now call functions to get directory listing, change directory, get and send files. Arg1 – pointer to buffer containing ftp server reply string (for example “226 logged in”) Arg2 – length of buffer containing ftp server reply string Arg3 - 0
HS_FTPCLI_USR_EV_CLOSED	FTP control connection and FTP session closed (error condition). Arg1 – 0 Arg2 – 0 Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes.
HS_FTPCLI_USR_EV_CONNFAIL	Outgoing TCP connection to remote FTP server failed to establish. Arg1 – pointer to null terminated additional information string Arg2 – 0 Arg3 – 0

HS_FTPCLI_USR_EV_SRVERR	FTP session closed due to server error reply. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes.
HS_FTPCLI_USR_EV_UNEXP	Unexpected server response, session closed. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes.
HS_FTPCLI_USR_EV_UNKNOWN	Unrecognized server response, session closed. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_NOCTX	FTP session closed, HS FTP library run out of free concurrent connection contexts Arg1 – 0 Arg2 – 0 Arg3 – 0
HS_FTPCLI_USR_EV_TIMEDOUT	FTP session closed due to timeout Arg1 – pointer to null terminated sting with additional information Arg2 – 0 Arg3 – 0
HS_FTPCLI_USR_EV_NOMEM	FTP session closed, no free memory Arg1 – 0 Arg2 – 0 Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_LIST	This event is used by HS FTP module to return a buffer containing current directory listing from remote FTP server after HsFtpCliList function call

	Arg1 – pointer to start of buffer containing remote FTP server current directory listing. Arg2 – length of buffer Arg3 – 0 User application must copy buffer content to local memory. On return from the callback the buffer memory is deallocated by HS FTP module.
HS_FTPCLI_USR_EV_CWD_FAILED	HsFtpCliChDir function failed (CWD FTP operation failed). Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_CWD_DONE	HsFtpCliChDir function successfully completed – Current directory on remote FTP server successfully changed to new specified directory. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_RXDONE	File successfully received – HsFtpCliGetFile function completed. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_TXDONE	File successfully transmitted – HsFtpCliSendFile completed. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_RXINCOMPL	File receive operation failed. Partial file received. Arg1 – 0 Arg2 – 0 Arg3 – 0
HS_FTPCLI_USR_EV_TX_STATUS	Notifies the application that the next block of file data has been transmitted Arg3 – size of data block that has been transmitted

HS_FTPCLI_USR_EV_RX_STATUS	Notifies the application that the next data block of file from remote FTP server has been received Arg3 – size of data block that has been received
HS_FTPCLI_USR_EV_TYP_FAILED	Setting transfer type failed (TYPE command failed) Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_STOR_FAILED	File transmit operation failed – HsFtpCliSendFile function completed with error. Arg1 – 0 Arg2 – 0 Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_ABORTED	Current FTP operation aborted – HsFtpCliAbort function completed Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_ABOR_FAILED	Abort operation failed - HsFtpCliAbort function completed with error Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – HS FTP library integer debug code – indicates from which state the callback function was called. Refer to section 3.3 for full list of codes
HS_FTPCLI_USR_EV_MKD_FAILED	HsFtpCliCreateDir function failed (MKD FTP operation failed). Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0

HS_FTPCLI_USR_EV_MKD_DONE	HsFtpCliCreateDir function successfully completed – Directory on remote FTP server successfully created. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_RMD_FAILED	HsFtpCliRemoveDir function failed (RMD FTP operation failed). Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_RMD_DONE	HsFtpCliRemoveDir function successfully completed – Directory on remote FTP server successfully deleted Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_DELE_DONE	HsFtpCliDeleteFile function successfully completed – File on remote FTP server successfully deleted Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_DELE_FAILED	HsFtpCliDeleteFile function failed (DELE FTP operation failed). Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_RENAME_DONE	HsFtpCliRename function successfully completed file or folder on remote FTP server renamed Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0

HS_FTPCLI_USR_EV_RENAME_FAILED	HsFtpCliRename function failed (RNFR FTP command failed) Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_PWD_DONE	HsFtpCliGetCurrentDirectory function completed successfully. Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_PWD_FAILED	HsFtpCliGetCurrentDirectory function failed (FTP PWD command failed). Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0
HS_FTPCLI_USR_EV_NOOP_DONE	Response to NOOP command received Arg1 – pointer to buffer containing ftp server reply string Arg2 – length of buffer containing ftp server reply string Arg3 – 0

2.2.2.5 Information Codes

HS_FTPC_ST_WAIT_WELCOME	- waiting for initial server reply on control session setup
HS_FTPC_ST_CHDIR_RSP	- waiting for reply to CWD command
HS_FTPC_ST_LIST_WAIT_RSP	- waiting for reply to LIST command
HS_FTPC_ST_WAIT_PASV_RSP	- waiting for reply to PASV command
HS_FTPC_ST_WAIT_USER_RSP	- waiting for reply to USER command
HS_FTPC_ST_WAIT_PASS_RSP	- waiting for reply to PASS command
HS_FTPC_ST_WAIT_LSTDATA_CONN	- waiting for data connection (LIST command)
HS_FTPC_ST_WAIT_CWDDATA_CONN	- waiting for data connection (CWD command)
HS_FTPC_ST_WAIT_TYPE_RSP	- waiting for reply to TYPE command
HS_FTPC_ST_WAIT_RETRDATA_CONN	- waiting for data connection (RETR command)
HS_FTPC_ST_WAIT_STORDATA_CONN	- waiting for data connection (STOR command)
HS_FTPC_ST_STOR_WAIT_RSP	- STOR wait response state
HS_FTPC_ST_RETR_WAIT_RSP	- RETR wait response state
HS_FTPC_ST_STOR_SENDING	- STOR sending file data state
HS_FTPC_ST_WAIT_ABOR_RSP	- waiting for reply to ABOR command
HS_FTPC_ST_LIST_WAIT_DTCLOSE	- waiting for data connection to close
HS_FTPC_ST_MKDIR_RSP	- waiting for response to MKD
HS_FTPC_ST_RMDIR_RSP	- waiting for response to RMD
HS_FTPC_ST_DELE_RSP	- waiting for response to DELE

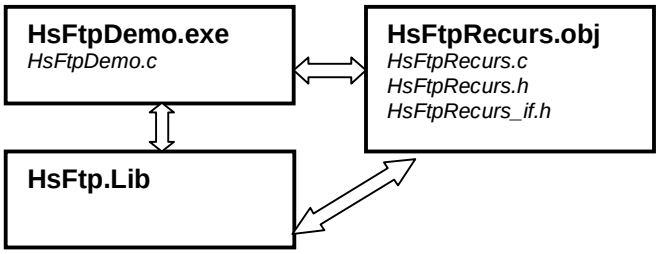
HS_FTPC_ST_WAIT_MKDDATA_CONN	- waiting for data connection (MKD)
HS_FTPC_ST_WAIT_RMDDATA_CONN	- waiting for data connection (RMD)
HS_FTPC_ST_WAIT_DELEDATA_CONN	- waiting for data connection (DELE)
HS_FTPC_ST_WAIT_NOOP_RSP	- waiting for response to NOOP
HS_FTPC_ST_WAIT_PWD_RSP	- waiting for response to PWD
HS_FTPC_ST_WAIT_RNFR_RSP	- waiting for response to RNFR
HS_FTPC_ST_WAIT_RNTO_RSP	- waiting for response to RNTO
HS_FTPC_ST_LIST_WAIT_DATA_CLOSED	- waiting for data connection closed in LIST sequence
HS_FTPC_ST_LOGGED_IN	- logged_in state

2.2.2.6 Recursive Folder Operations

HS FTP Source Code Library package includes recursive folder operations module HsFtpRecurs which includes the following functions:

- HsFtpRecursDownloadFolder – recursively download entire folder with all sub-folders and files from remote FTP server.
- HsFtpRecursUploadFolder – recursively upload entire folder with all sub-folders and files to remote FTP server.
- HsFtpRecursDeleteFolder – recursively delete entire folder with all sub-folders and files from remote FTP server.

Recursive operations module is not part of the code HSFTP library, but instead is implemented as an object module that links to the main application which in turn links in the code HSFTP.LIB



2.2.2.6.1 API Functions

2.2.2.6.1.1 HsFtpRecursInit

Declaration:
Extern void HsFtpRecursInit (hsftp_recursive_callback_t *cb)

Summary:
This function is used to initialise recursive folder operations module, it must be used before any other functions of this module are used

Parameters:

hsftp_recursive_callback_t *cb - Function pointer to recursive module callback functions which receives event notifications of folder operations completion, failures, status and progress

Return values:

None

Sample usage:

HsFtpRecursInit(hsftp_recursive_callback);

2.2.2.6.1.2 HsFtpRecurseCleanUp

Declaration:
Extern void HsFtpRecurseCleanUp (void)

Summary:
This function is used to release all resources used by recursive folder operations module

Parameters:

None

Return values:

None

Sample usage:

HsFtpRecurseCleanUp();

2.2.2.6.1.3 HsFtpRecursTick

Declaration:
void HsFtpRecursTick(void)

Summary:
This function must be called as often as possible from the main program loop.

Parameters:
None

Return values:

None

Sample usage:

```
while (1)
{
        if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
        {
                if (GetMessage(&msg, NULL, 0, 0))
                {
                        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                        {
```

```
                                        TranslateMessage(&msg);
                                        DispatchMessage(&msg);
                                }
                        }
                        else
                        {
                                break;
                        }
                }
                else
                {
                         HsFtpTick();
                        HsFtpRecursTick();

                        Sleep(1);

                        update_stats();
                }
        }
```

2.2.2.6.1.4 HsFtpRecursDownloadFolder

Declaration:
extern int HsFtpRecursDownloadFolder (long session_handle, unsigned char *foldername, int overwrite);

Summary:
This function is used to download an entire folder with all sub-folders and files from remote FTP server

Parameters:
long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char *foldername - Folder name to download, null terminated string

int overwrite - 1 = Any files already existing at local file system shall be overwritten
0 = Files already existing at local files system shall be overwritten only if the files size does not match the files size of the corresponding file at the remote FTP server, otherwise if the file sizes match, the file is not downloaded

Return values:

Value	Description
HS_FTP_RC_OK	**Success,** , Recursive folder download started. The actual result shall be asynchronously indicated via user callback event
HSFTP_RECURS_RC_NOTINIT	Recursive folder operations module not initialised
HSFTP_RECURS_RC_INVPAR	invalid parameters supplied
HSFTP_RECURS_RC_BUSY	command is invalid for current session state

Sample usage:

rc = HsFtpRecursDownloadFolder(ftp_session, editstr, overwrite);

2.2.2.6.1.5 HsFtpRecursUploadFolder

Declaration:
extern int HsFtpRecursUploadFolder (long session_handle, unsigned char *foldername)

Summary:
This function is used to upload an entire folder with all sub-folders and files to remote FTP server

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char *foldername - Folder name to upload, null terminated string

Return values:

Value	Description
HSFTP_RECURS_RC_OK	**Success,** Recursive folder upload started. The actual result shall be asynchronously indicated via user callback event
HSFTP_RECURS_RC_NOTINIT	Recursive folder operations module not initialised
HSFTP_RECURS_RC_INVPAR	invalid parameters supplied
HSFTP_RECURS_RC_BUSY	command is invalid for current session state

Sample usage:

rc = HsFtpRecursUploadFolder(ftp_session, editstr);

2.2.2.6.1.6 HsFtpRecursDeleteFolder

Declaration:
extern int HsFtpRecursUploadFolder (long session_handle, unsigned char *foldername)

Summary:
This function is used to delete an entire folder with all sub-folders and files from remote FTP server

Parameters:

long session_handle - HS FTP session handle, returned in HsFtpCliConnect call.

unsigned char *foldername - Folder name to delete, null terminated string

Return values:

Value	Description
HSFTP_RECURS_RC_OK	**Success,** Recursive folder deletion started. The actual result shall be asynchronously indicated via user callback event
HSFTP_RECURS_RC_NOTINIT	Recursive folder operations module not initialised
HSFTP_RECURS_RC_INVPAR	invalid parameters supplied
HSFTP_RECURS_RC_BUSY	command is invalid for current session state

Sample usage:

rc = HsFtpRecursDeleteFolder(ftp_session, editstr);

2.2.2.6.2 Recursive Operations Callback and Events

2.2.2.6.2.1 Event Callback Prototype

typedef void hsftp_recursive_callback_t (long ftp_session, int ev, long arg1, long arg2);

Parameter	Description
long ftp_session	FTP session handle.
int ev	Event id, the full list and description is provided in next section
long arg1	Parameter 1, specific to event id
long arg2	Parameter 2, specific to event id

2.2.2.6.2.2 Events

Event	Description
HSFTP_RECURS_USR_EV_FOLDER_DOWNLOAD_FAILED	Recursive folder download operation failed. Arg1 – Recursive operations module return code, see next section for a list of return codes Arg2 – Core HS FTP library event that caused this event, see section 3.2 for a list of core HS FTP events
HSFTP_RECURS_USR_EV_FOLDER_DOWNLOAD_DONE	Recursive folder download operation complete with success. Arg1 – 0 Arg2 – 0
HSFTP_RECURS_USR_EV_FOLDER_DOWNLOAD_STATUS	reports start of stop of individual file download operation Arg1 – If Arg2 == 1, long pointer to a pathname of the item now starting to download Arg2 – status code: 1 = Start of individual item download 2 = Completion of individual item download
HSFTP_RECURS_USR_EV_FOLDER_UPLOAD_FAILED	Recursive folder upload

	operation failed. Arg1 – Recursive operations module return code, see next section for a list of return codes Arg2 – Core HS FTP library event that caused this event, see section 3.2 for a list of core HS FTP events
HSFTP_RECURS_USR_EV_FOLDER_UPLOAD_DONE	Recursive folder upload operation complete with success. Arg1 – 0 Arg2 – 0
HSFTP_RECURS_USR_EV_FOLDER_UPLOAD_STATUS	reports start of stop of individual file upload operation Arg1 – If Arg2 == 1, long pointer to a pathname of the item now starting to upload Arg2 – status code: 1 = Start of individual item upload 2 = Completion of individual item upload
HSFTP_RECURS_USR_EV_FOLDER_DELETE_FAILED	Recursive folder delete operation failed. Arg1 – Recursive operations module return code, see next section for a list of return codes Arg2 – Core HS FTP library event that caused this event, see section 3.2 for a list of core HS FTP events
HSFTP_RECURS_USR_EV_FOLDER_DELETE_DONE	Recursive folder delete operation complete with success. Arg1 – 0 Arg2 – 0
HSFTP_RECURS_USR_EV_FOLDER_DELETE_STATUS	reports start of stop of individual item delete operation Arg1 – If Arg2 == 1, long pointer to a pathname of the item now being deleted

	Arg2 – status code: 1 = Start of individual item delete 2 = Completion of individual item delete
HSFTP_RECURS_USR_EV_FOLDER_PROGRESS	Indicates individual item operation progress (item upload, download or delete) Arg1 – percent complete 0 – 100 Arg2 – 0

2.2.2.6.3 Recursive Operations Module Return Codes

Return code	**Description**
HSFTP_RECURS_RC_OK	Success
HSFTP_RECURS_RC_BUSY	Invalid state, recursive folder operation in progress
HSFTP_RECURS_RC_NOTINIT	HsFtpRecurs module not initialised
HSFTP_RECURS_RC_INVPAR	Invalid parameters
HSFTP_RECURS_RC_NOMEM	Out of memory
HSFTP_RECURS_RC_RCHDIR	Remote change directory failed
HSFTP_RECURS_RC_LIST	List command failed
HSFTP_RECURS_RC_FILE	Individual file operation failed
HSFTP_RECURS_RC_RMDIR	Remove directory at remote FTP server failed
HSFTP_RECURS_RC_LCHDIR	Failed to change into local directory
HSFTP_RECURS_RC_FOPEN	Failed to open local file
HSFTP_RECURS_RC_RMKDIR	Failed to create folder at remote FTP server
HSFTP_RECURS_RC_FSEND	Failure during sending a file
HSFTP_RECURS_RC_PATH	Pathname too long

2.2.3 HsSmtp

2.2.3.1 Overwiew

HS SMTP implements the client side of Simple Mail Transfer Protocol (SMTP) over TCP socket layer according to RFC 821. Support for transfer of basic message header and text is provided.

HS SMTP upports ESMTP extension for LOGIN Authentication using Base64 encoding and message sending to multiple recipients from address list.

HS SMTP supports sending binary file attachments using MIME version 1.0 base64 encoding

HS SMTP Library incorporates the necessary state machine, transparency procedures, and server response processing required to provide for a simple and robust SMTP client implementation

2.2.3.2 HsSmtp API

2.2.3.2.1 HsSmtpInit

Declaration:
extern int HsSmtpInit(hssmtp_init_t *init);

Summary:

This function initialises HS SMTP Library. It must be called at initialisation, before any other functions are called.

Calling HsSmtpInit twice will return an error. You can call HsSmtpDestroy first to de-initialise HS SMTP Library and then call HsSmtpInit again

Parameters:
hssmtp_init_t *init – pointer to initialization structure of type hssmtp_init_t. This structure is described as follows:

Parameter name	Description
***start_timer**	**Pointer to start timer callback function. HSSMTP calls this function whenever it needs to start a timer** **Start Timer callback function is defined as:** typedef long hssmtp_timer_start_t(unsigned long timeout_ms, void *arg, hssmtp_timer_cb_t *cb); **timeout_ms – timeout value in milliseconds** **arg – void parameter argument, must be passed back in callback function cb**

	cb – callback function to be called when timer expires: defined as typedef void hssmtp_timer_cb_t(void *arg);
***stop_timer**	**Pointer to stop timer callback function. HSSMTP calls this function whenever it needs to stop a previously started timer.** **Stop timer function has the following prototype:** **typedef void hssmtp_timer_stop_t(long timer_han);** **timer_han – timer handle**
create_file	**Pointer to create file callback function. HSSMTP calls this function when it needs to open a disk file for reading or for writing.** **It has the following prototype:** void *hssmtp_create_file_t(unsigned char *filename, int mode, int open_always); **filename – name of file to open** **mode – access mode:** **HSSMTP_FILE_READ – read access** **HSSMTP_FILE_WRITE – write access** **open_always: 1 = Open file always regardless of file existed before or not; 0=normal operation** **returns file handle cast to void pointer.**
close_file	**Pointer to function callback to close a file. HSSMTP calls this function when it needs to close a file.** **It has the following prototype:** **void hssmtp_close_file_t(void *hFile);** **hFile – file handle (obtained with create_file call)**
write_file	**Pointer to function callback to write file data. HSSMTP calls this function whenever it needs to save next block of data to an open file.** **It has the following prototype:** int hssmtp_write_file_t(void *hFile, void *pData, int length, int *bytes_written); hFile – file handle pData – pointer to data buffer to write int length – length of data buffer in bytes to write bytes_written – pointer to integer to receive actual number of bytes written to file returns 1: operation successful; 0: file write error occurred

read_file	**Pointer to function callback to read file data. HSSMTP calls this function whenever it needs to read next block of data from an open file.** **It has the following prototype:** int hssmtp_read_file_t(void *hFile, void *pData, int length, int *bytes_read); hFile – file handle pData – pointer to data buffer to read into int length – length of data buffer in bytes to read bytes_read – pointer to integer to receive actual number of bytes read returns 1: operation successful; 0: file write error occurred
getticks	**Pointer to function callback to obtain current number of millisecond ticks since system boot-up.** **It has the following prototype:** **typedef unsigned long hssmtp_get_ticks_fn_t(void);** **returns number of milliseconds since bootup**
unsigned char	**domain[HS_SMTP_MAX_DOMAIN]; // local domain name** **Network name identifying this host. Hssmtp will use this name when communicating to SMTP server. It can be set to any arbitrary string, such as "my desktop".**

Return values:

Value	Description
HS_SMTP_RC_OK	Success
HS_SMTP_RC_ALRINIT	HS SMTP Library is already initialised
HS_SMTP_RC_INV_PAR	Invalid parameter(s)

Sample usage:

```
hssmtp_init_t smtp_init = {0};

        strcpy(smtp_init.domain, "mydomain");
        smtp_init.start_timer = sock_timer_start;
        smtp_init.stop_timer = sock_timer_stop;

        smtp_init.create_file = hssmtp_create_file;
        smtp_init.close_file = hssmtp_close_file;
        smtp_init.read_file = hssmtp_read_file;
        smtp_init.write_file = hssmtp_write_file;
        smtp_init.getticks = hs_sock_get_ticks;
        smtp_init.getfile_size = hssmtp_get_filesize;
```

```
if (HsSmtpInit(&smtp_init) != HS_SMTP_RC_OK)
{
        MessageBox(0,"HS SMTP Initialisation failure", 0, 0);
        return;
}
```

2.2.3.2.2 HsSmtpDestroy

Declaration:
int HsSmtpDestroy(void);

Summary:
This function de-initialises HS SMTP Library, releases all used resources and cleans up Socket Interface Layer.

Parameters:
None

Return values:

Value	Description
HS_SMTP_RC_OK	Success
HS_SMTP_RC_NOTINIT	HS SMTP Library not initialised

Sample usage:

HsSmtpDestroy();

2.2.3.2.3 HsSmtpTick

Declaration:
int HsSmtpTick (void);

Summary:
This function must be called from the user application periodically and as often as possible. This function drives internal operation of the socket layer (reading events from TCP sockets) and pacing internal SMTP timers.

Parameters:
None

Return values:

Value	Description
HS_SMTP_RC_OK	Success
HS_SMTP_RC_NOTINIT	HS SMTP Library not initialised

Sample usage:

```
// main message processing loop
while (1)
{
	if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
	{
		if (GetMessage(&msg, NULL, 0, 0))
		{
			if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
			{
				TranslateMessage(&msg);
				DispatchMessage(&msg);
			}
		}
		else
		{
			break;
		}
	}
	else
	{
		HsSmtpTick();

		Sleep(1);

		if (adhandle)
		{
			r = pcap_next_ex( adhandle, &header, &pkt_data);
			if (r > 0)
			{
				HsIpReceivePacket(pkt_data, (int)header->caplen);
			}
		}
	}
}
```

2.2.3.2.4 HsSmtpSendMail

Declaration:

Int **HsSmptSendMai**l(hs_smtp_mail_t *m, long *session_handle)

Summary:

This function sends an email text message to remote SMTP server according to RFC821. Fill out hs_smtp_mail_t structure members and set pointer to storage for SMTP session handle prior to calling the function. Various structure fields are explained below. The function is non-blocking and complete asynchronously. The completion of mail transfer operation is reported via event callback function. The pointer to callback function is also specified in hs_smtp_mail_t structure.

Parameters:

hs_smtp_mail_t *m - Pointer to parameter structure as follows:

Field name	Description
srv_name	Pointer to SMTP server name Max. length limited to 80 bytes (Note 1)
srv_ip	Pointer to SMTP IP address string in the format “n.n.n.n” (Note 1)
srv_port	Decimal SMTP server port number. (usually 25)
send_addr	Sender email address string Max. length limited to 80 bytes
recv_addr	Destination email address string Max. length limited to 80 bytes
subj	Message subject Max. length limited to 80 bytes
cc	Cc email address Max. length limited to 80 bytes
msgp	Pointer to message text buffer
msglen	message text buffer length
do_auth	Integer. Set to 1 if SMTP server requires authenticaton. LOGIN method authentication of ESMTP shall be performed
usrname	Pointer to username string for authenticaton, ignored if do_auth is 0
password	Pointer to password string for authentication, ignored if do_auth is 0
send_file	1=send file attachment, 0=no file attachment
filename	filename of the file to send, valid only if sen_file = 1
callback	Pointer to callback function in user code to receive completion and error event notifications from HS SMTP Library. (See section 3 for more details)
session_ref	User data associated with this session.This parameter is passed to the callback function in used code without modification
use_latin1	Integer flag which switches on the use of iso-8859-1 encoding. for characters from 0x7f to 0xff use escape sequence ==<hex code> Set this flag to TRUE when sending file attachments. Set to false when sending mail without attachments.

Note 1: Either server name or server IP address string can be supplied to the function. If srv_ip is non NULL, srv_ip string shall be used to connect to SMTP server. If srv_ip is null, srv_name string shall be used to resolve server IP address first before connecting to it.

long *session_handle - HS SMTP fills in the variable pointed to by this parameter with SMTP session reference. Use this parameter to close a session with HsSmtpAbortMail if required.

Return values:

Value	Description
HS_SMTP_RC_OK	Success

HS_SMTP_RC_NOTINIT	HS SMTP Library not initialized
HS_SMTP_RC_INV_PAR	Invalid parameters specified
HS_SMTP_RC_NO_CTX	No free contexts (Maximum 5 concurrent sessions supported)
HS_SMTP_RC_NONAME	Socket layer cannot resolve SMTP server name
HS_SMTP_RC_TCPCONNFAIL	Socket layer failed to connect to SMTP server

Sample usage:

rc = HsSmtpSendMail(&m, &session_handle);

2.2.3.2.5 HsSmptAbortMail

Declaration:
int **HsSmptAbortMail**(long session_handle);

Summary:
This function is used to abort current mail transfer already in progress. The function returns immediately, but the clean mail session termination completes asynchronously and the result is reported via event callback function.

Parameters:

long session_handle - SMTP session handle (returned with HsSmtpSendMail)

Return values:

Value	Description
HS_SMTP_RC_OK	Success
HS_SMTP_RC_NOTINIT	HS SMTP Library not initialized
HS_SMTP_RC_INV_PAR	Invalid parameters specified
HS_SMTP_RC_NOT_OPEN	mail session not open

Sample usage:

HsSmptAbortMail(session_handle);

2.2.3.3 HS SMTP to User Event Callback and Events

2.2.3.3.1 Event Callback Prototype

typedef int **smtp_callback_t**(void *session_ref, int ev, long arg1, long arg2);

Parameter	Description
*session_ref	User data, passed unchanged from the call to HsSmtpSendMail
ev	Event code (see event code table below)
Arg1	Parameter 1 (dependant on event code)
Arg2	Parameter 2 (dependant on event code)

2.2.3.3.2 Event Codes

Event	Arg1	Arg2	Description
HS_SMTP_USR_EV_DONE	0	0	Mail transfer complete, mail has been successfully sent to SMTP server and mail session is terminated
HS_SMTP_USR_EV_CLOSED	0	0	TCP session closed by remote peer
HS_SOCK_EV_CONN_FAILED	0	0	TCP connect attempt failed
HS_SMTP_USR_EV_SRVERR	Unsigned char *buffer pointer to server reply string	Int length length of server reply string	Mail session closed because SMTP server returned error. The error string returned by the server is passed in parameters arg1 and arg2
HS_SMTP_USR_EV_TIMEDOUT	Unsigned char *error Null terminated error string	0	Mail session closed because current protocol operation timed out. Arg1 parameter has pointer to error string with additional information about the timeout condition.
HS_SMTP_USR_EV_PROGRESS	Integer number from 0 to 100 representing percentage of transmitted message text	0	Progress indication, parameter 1 has a number from 1 to 100, representing percentage of message text sent so far.
HS_SMTP_USR_EV_PROGRESS_F	Integer number from 0 to 100 representing percentage of file attachment transmitted so far	0	Progress indication, parameter 1 has a number from 1 to 100, representing percentage of of file attachment

			transmitted so far.
HS_SMTP_USR_EV_GETNEXT	Pointer to hs_smtp_mail_t message structure	0	When a message has been successfully transmitted to server, the callback function is called with this primitive. If the user application wishes to send the same message to another recipient, it should set the new *recv_addr in the hs_smtp_mail_t structure and return TRUE. If the user application returns FALSE, HS SMTP will exit the mail session

2.2.4 HsPop3

2.2.4.1 Overview

HS POP3 implements the client side of Post Office Protocol Version 3 (POP3) over TCP socket layer according to RFC 1939. Among other features, the library supports user authentication, reception of basic internet headers and text, message deletion and statistics.

HS POP3 Library incorporates the necessary state machine, transparency procedures, and server response processing required to comply to a simple and robust POP3 client implementation.

2.2.4.2 HsPop3 API

2.2.4.2.1 HsPop3Init

Declaration:
int **HsPop3Init**(hs_pop3_api_t *api);

Summary:
This function initialises HS POP3 Library. It must be called at initialisation, before any other functions are called.

Calling HsPop3Init twice will return an error. You can call HsPop3Destroy first to deinitialise HS Pop3 Library and then call HsPop3Init again

Parameters:
hs_pop3_api_t *api - Pointer to parameter structure, please see next section for details

Return values:
- *HS_POP3_RC_ALRINIT – HS POP3 Library already initialised*
- *HS_POP3_RC_INV_PAR – invalid parameters specified*
- *HS_POP3_RC_SOCKINIT_FAIL – socket layer initialisation failed*
- *HS_POP3_RC_OK - success*

Sample usage:

```
api.start_timer = sock_timer_start;
api.stop_timer = sock_timer_stop;
api.event_cb = pop3_callback;

rc = HsPop3Init(&api);
```

2.2.4.2.1.1 Initialisation Structure Definition (hs_pop3_api_t)

Field name	Description
start_timer_t *start_timer;	Pointer function in user code used by HS POP3 to start a timer Prototype:

	long start_timer_t(unsigned long timeout_ms, void *arg, timer_callback_t *cb);
	
	Parameters:
	Handle – POP3 session handle
	timeout_ms – timeout in milliseconds
	*cb – pointer to function in HS POP3 code that the user code should call on timer expiry.
	
	arg – argument that HS POP3 passed to start timer callback. This argument must be passed back unchanged to timer expiry callback.
	
	Callback function prototype:
	typedef void stop_timer_t(long timer_han);
	
	timer_han – timer handle
	
	Returns:
	timer handle or NULL if timer start error

stop_timer_t *stop_timer	Pointer function in user code used by HS POP3 to stop a timer Prototype void stop_timer_t(long timer_handle); Parameters: timer_handle – timer handle (returned from call to start_timer)
event_callback_t *event_cb	Pointer function in user code used by HS POP3 to report operation results, status and errors.

2.2.4.2.2 HsPop3Destroy

Declaration:
int **HsPop3Destroy**(void);

Summary:
This function de-initialises HS POP3 Library and releases all used.

Parameters:
None

Return values:
- *HS_POP3_RC_OK – success, HS POP3 Library successfully de-initialised*
- *HS_POP3_RC_NOTINIT – cannot destroy, HS POP3 Library not yet initialised*

Sample usage:

```
HsPop3Destroy();
```

2.2.4.2.3 HsPop3GetMail

Declaration:
int HsPop3GetMail(hs_pop3_session_t *s, long *session_handle);

Summary:
This function initiates mail reception from POP3 server. It works as follows:
1) HS POP3 contacts POP3 server with login information
2) If login is authorised, HS POP3 checks to see if maibox has any messages
3) If mailbox is not empty, HS POP3 follows this procedure for each message until all messages processed:

A) Get message id from server and report it to user application via HS_POP3_USR_EV_MSGID event

B) If user application indicates via return of callback that it wants to receive this message, HS POP3 receives
message header, parses header fields and passes it to application via HS_POP3_USR_EV_DONE_MSGHDR event.

C) HS POP3 then receives the message body from the server block by block and each block's data is returned
to application via HS_POP3_USR_EV_DONE_MSGBLK event.

When message is fully received and all blocks have been returned to application, HS POP3 generates
HS_POP3_USR_EV_DONE_MSG event

D) If there are more messages to be received from the mail server, HS POP3 does to step A

E) When all messages are processed, HS POP3 calls event callback with HS_POP3_USR_EV_DONE event
passing total number of messages in mailbox and number of received messages

Depending on parameters, HS POP3 can delete messages from the server after successful reception..

Parameters:
hs_pop3_session_t *s - Pointer to session parameters structure:

srv_name – POP3 server name (Note 1)
username – POP3 username, 0 terminated
password – POP3 password, 0 terminated
srv_port - POP3 server port (usually 110)
delete_msgs – is TRUE, delete messages from server
user_data – user handle, not modified by HS POP3 and always passed back in event callback function.

Note 1: Either server name or server IP address string can be supplied in the srv_name to the function

long *session_handle - HS POP3 fills in the variable pointed to by this parameter with POP3 session reference. Use this parameter to close a session with HsPop3Abort if required.

Return values:
- *HS_POP3_RC_NOTINIT – HS POP3 Library not initialised*

- *HS_POP3_RC_INV_PAR – invalid parameters specified*
- *HS_POP3_RC_NOPASS – no password specified in parameters*
- *HS_POP3_RC_NOUSER – no username specified in parameters*
- *HS_POP3_RC_NO_CTX – maximum number of open HS POP3 sessions reached*
- *HS_POP3_RC_NONAME – POP3 server name could not be resolved*
- *HS_POP3_RC_TCPCONNFAIL- TCP connection to server could not be established*
- *HS_POP3_RC_TIMERFAIL – timer start failure*
- *HS_POP3_RC_OK – success*

Sample usage:

rc = HsPop3GetMail(&ss, &session_handle);

2.2.4.2.4 HsPop3Abort

Declaration:
Int **HsPop3Abort**(long session_handle);

Summary:
This function is used to abort current mail session in progress. HS POP3 library will discard current message (if not fully received) free message memory and exit mail session cleanly via POP3 QUIT command. This means that the session is not closed immediately, but only after reception of a valid response to QUIT command or timeout.

Parameters:
long session_handle - POP3 session handle of the session to abort

Return values:
- *HS_POP3_RC_OK – success*
- *HS_POP3_RC_NOTINIT – HS POP3 Library not initialized*
- *HS_POP3_RC_INV_PAR – invalid parameters specified*

Sample usage:

HsPop3Abort(session_handle);

2.2.4.2.5 HsPop3GetErrStr

Declaration:
unsigned char *HsPop3GetErrStr (int rc);

Summary:
This function is used to convert integer HS POP3 return code into a readable string – error description.

Parameters:
int rc - HS POP3 integer return code

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

Sample usage:

```
rc = HsPop3Abort(session_handle);
if (rc != HS_POP3_RC_OK)
{
        unsigned char s[200];
        sprintf(s, "ERROR: HS POP3 ERROR (%s)", HsPop3GetErrStr(rc));
        color = RED;
        SetDlgItemText(dlg_main, IDC_STATIC_STATUS, s);
        break;
}
```

2.2.4.3 HS POP3 to USER Event Callback and Events

2.2.4.3.1 Event Callback Prototype

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

Parameter	Description
handle	User handle, the same as specified in call to HsPop3getMail in user_data parameter of hs_pop3_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 POP3. Where HS POP3 needs a specific return, it is clearly specified in this document

2.2.4.3.2 Events

Event	Arg1	Arg2	Description
HS_POP3_USR_EV_MSGID	Pointer to message ID string, zero terminated with maximum length 71 bytes (not including last zero)	0	User application at this point may go through the list of locally stored (previously received) messages to see if it already has a message with the same message id. If the user wishes to receive this message, the

			return from callback function must be TRUE, otherwise to skip the message return FALSE
HS_POP3_USR_EV_DONE_MSGHDR	Pointer to message header structure received and parsed by HS POP3. The message header structure is defined in hs_pop3_msg_t	0	User application should store the message header of the message which is going now to be received by HS POP3 User must copy the passed structure content into a local structure. HS POP3 shall release the internally allocated memory for the structure pointer on return from callback function
HS_POP3_USR_EV_DONE_MSGBLK	Pointer to buffer next block of message data received	Length of data block received	Next block of message data received User must copy the passed buffer content into a local message storage (or mailbox file). HS POP3 shall release the internally allocated memory for the buffer pointer on return from callback

			function
HS_POP3_USR_EV_DONE_MSG	0	0	Current message fully received
HS_POP3_USR_EV_DONE	Long number of messages received	0	Mail session complete
HS_POP3_USR_EV_CLOSED	0	0	Socket layer closed TCP connection. HS POP3 will release any allocated memory for a message being currently received
HS_POP3_USR_EV_CONNFAIL	0	0	HS POP3 could not connect to POP3 server
HS_POP3_USR_EV_SRVERR	Unsigned char pointer to server reply string	Length of server reply string	Session closed because of POP3 error response received from server HS POP3 will release any allocated memory for a message being currently received
HS_POP3_USR_EV_TIMEDOUT	Unsigned char pointer to additional information string (about the context in which timeout occurred)	Length of additional information string	Timed out waiting for server response, session closed. HS POP3 will release any allocated memory for a message being currently received
HS_POP3_USR_EV_PROGRESS1			HS POP3 reports number

			of messages in mailbox and total size of mailbox in octets. This event is indicated once per POP3 session
HS_POP3_USR_EV_PROGRESS2	Current message number (long)	Number of bytes received so far (long)	current receiving message number and number of bytes received so far. This event is indicated for each message block received until full message is received. This event is indicated for each message within the same session. It can be used to indicated individual message reception progress.
HS_POP3_USR_EV_INTERR	Pointer to zero terminated error string	0	Internal error occurred, the TCP POP3 sessoin has been closed and mail session context de-allocated

2.2.4.3.3 Message structure (hs_pop3_msg_t)

Data Member	Description
unsigned char from[HS_POP3_MAX_PATH];	FROM internet address, parsed out from message header
unsigned char date[HS_POP3_MAX_DATE];	DATE parsed out from message header

unsigned char subj[HS_POP3_MAX_SUBJ];	SUBJECT parsed out from message header
unsigned char msgid[HS_POP3_MAX_MSGID];	Message ID string as received from POP3 server for that message
int hdrlen;	Length of internet headers. The headers start from the first byte of the message
DWORD dwMsgLen;	Full message length including internet headers and data

2.2.5 HsNtp

2.2.5.1 Overview

HS NTP 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.

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

2.2.5.2 HsNtp API

2.2.5.2.1 HsNtpInit

Declaration:
int HsNtpInit (hs_ntp_api_t *api);**;**

Summary:
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:
hs_ntp_api_t *api - Pointer to parameter structure, please see next section for details

Return values:
- *HS_NTP_RC_ALRINIT – HS NTP Library already initialized*
- *HS_NTP_RC_INV_PAR – invalid parameters specified*
- *HS_NTP_RC_OK - Success*

Sample usage:

```
api.start_timer = start_timer;
api.stop_timer = stop_timer;
api.event_cb = ntp_callback;

rc = HsNtpInit(&api);
```

2.2.5.2.1.1 Initialisation Structure Definition (hs_ntp_api_t)

Field name	Description

start_timer_t *start_timer;	Pointer function in user code used by HS NTP to start a timer
	
	Prototype:
	long start_timer_t(lunsigned long secs, timer_callback_t *callback);
	
	Parameters:
	secs – timeout in seconds
	*callback – pointer to function in HS NTP code that the user code should call on timer expiry.
	
	Callback function prototype:
	void timer_callback_t(void);
	
	Returns:
	timer handle or NULL if timer start error

stop_timer_t *stop_timer	Pointer function in user code used by HS NTP to stop a timer Prototype void stop_timer_t(long timer_handle); Parameters: Timer_handle – timer handle (returned from call to start_timer)
event_callback_t *event_cb	Pointer function in user code used by HS NTP to report operation results, status and errors.

2.2.5.2.2 HsNtpDestroy

Declaration:
int **HsNtpDestroy**(void);

Summary:
*This function de-initialises HS Ntp Library***.**

Parameters:
None

Return values:
- *HS_NTP_RC_OK – success, HS NTP Library successfully de-initialised*
- *HS_ NTP _RC_NOTINIT – cannot destroy, HS NTP Library not yet initialized*

Sample usage:

HsNtpDestroy();

2.2.5.2.3 HsNtpGetErrStr

Declaration:
unsigned char *HsNtpGetErrStr (int rc);

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

Parameters:
int rc - HS NTP integer return code

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

Sample usage:

rc = (int)arg1;
sprintf(s, "Invalid Server Reply: (rc=%d): %s", rc, HsNtpGetErrStr(rc));

2.2.5.2.4 HsNtpGetTime

Declaration:
int HsNtpGetTime(hs_ntp_session_t *s);

Summary:
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:
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.

Return values:

- *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_SOCK_OPEN – Error opening UDP socket*
- *HS_NTP_RC_UDP_SOCK_SEND – Error sending UDP packet*
- *HS_NTP_RC_OK - Success*

Sample usage:

```
hs_ntp_session_t        ss = {0};

GetDlgItemText(hDlg, IDC_EDIT_SRVNAME, ss.srv_name, (sizeof ss.srv_name) - 1);

GetDlgItemText(hDlg, IDC_EDIT_PORT, port_str, sizeof port_str);

ss.source_port = (unsigned short)atoi(port_str);
rc = HsNtpGetTime(&ss);
```

2.2.5.3 HS NTP to USER Event Callback and Events

2.2.5.3.1 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

2.2.5.3.2 Events

Event	Arg1	Arg2	Description
HS_NTP_USR_EV_SRVRESP	Pointer to structure hs_ntp_info_t. See details in	0	Time request completed

	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.		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.

2.2.5.3.3 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 substract 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

2.2.6 HsDns

2.2.6.1 Overview

HsDns module implements client side of domain name resolution to an IP address. It is implemented using a simple client server request response model. HsDns provides services to all network level protocols.

2.2.6.2 HsDns API

2.2.6.2.1 HsDnsInit

Declaration:
extern int HsDnsInit(hsdns_cfg_t *cfg);

Summary:
This function initializes HsDns library and must be called once before any other functions are called.

Currently HsDns is initialized from HsSock module when user calls HsSockInit.

Parameters:
hsdns_cfg_t *cfg – pointer to structure containing configuration parameters.

Definition of hsdns_cfg_t structure

Data member	Description
int cache_size	Number of resolved DNS entries to keep in cache for fast lookup. When a user call function HsDnsGetIpbyName, HsDns first checks the cache of previously resolved DNS names to lookup the IP address. If the requested name is not in cache, then HsDns contacts DNS server and uses DNS resolution protocol to obtain the IP address
int num_requests	Number of concurrent DNS sessions to support.
unsigned long server_ip	32 bit DNS server ip address. If using static IP address, the user may set this value to DNS IP address, otherwise if using DHCP, set to 0. DNS IP address can be communicated to HsDns at the time when it is obtained from DHCP server. In this case the DNS IP address is set with function HsDnsSetParams.
hs_dns_timer_start_t *timer_start	Timer start callback function pointer. HsDns will use this function when it needs to start a timer Prototype: typedef long hs_dns_timer_start_t(unsigned long timeout_ms, void *arg, hs_dns_timer_cb_t *cb); timeout_ms – timeout in milliseconds arg – argument passed from HsDns cb – User code must call this function pointer callback on timer

	expiry Timer expiry function prototype: typedef void hs_dns_timer_cb_t(void *arg); The timer_start function returns long timer handle arg – same argument as passed to timer_start function
hs_dns_timer_stop_t *timer_stop	Timer start callback function pointer. HsDns will use this function when it needs to stop a timer Prototype: typedef void hs_dns_timer_stop_t(long timer_han); timer_han – timer handle obtained with call to timer_start.

Return values:

- HS_DNS_RC_BAD_PARAMS – Invalid parameter(s)
- HS_DNS_RC_ALR_INIT – Library already initialized
- HS_DNS_RC_OK - Success

Sample usage:

```
dns_cfg.cache_size = 3;
dns_cfg.num_requests = 3;
dns_cfg.server_ip = dns_server_ip;
dns_cfg.timer_start = init->timer_start_fn;
dns_cfg.timer_stop = init->timer_stop_fn;

HsDnsInit(&dns_cfg);
```

2.2.6.2.2 HsDnsCleanUp

Declaration:
extern void HsDnsCleanUp(void);

Summary:
De-allocates resources and closes HsDns services.

Currently this function is called from HsSock module, when user calls HsSockCleanUp();

Parameters:
None

Return values:
None

Sample usage:

HsDnsCleanUp();

2.2.6.2.3 HsDnsSetParams

Declaration:
extern void HsDnsSetParams(hsdns_params_t *pPar);

Summary:
Sets HsDns configuration parameters. This function can be called at any time after HsDnsInit function already has been called.

If DHCP is used, this function is currently called by HsSock module when HsDhcp modifies HsSock that is has obtained IP parameters from DHCP server, including DNS server IP address. HsSock sets new DNS server IP address in hsdns_params_t structure.

Parameters:
hsdns_params_t *pPar – parameter structure, defined as follows:

Data member	Description
unsigned long server_ip;	**DNS server ip address**

Return values:
None

Sample usage:

The following function is part of HsSock module. HsSock module automatically handles obtaining new IP parameters for HsTCPIPv4 stack using DHCP (if enabled). The code shown below is HsDhcp callback function in HsSock code, which gets called when HsDhcp has obtained IP parameters from DHCP server:

```
/*
 * DHCP event callback
 */
static void hssock_dhcp_event_callback(int ev, long arg1, long arg2)
{
	hsip_params_t sPar = {0};
	hs_tcp_params_t sTcpPar = {0};
	hs_udp_params_t sUdpPar = {0};
	hs_arp_params_t sArpPar = {0};
	hsdns_params_t sDnsPar = {0};
	char str[256];
	char ipaddrstr[17] = {0};

	hs_dhcp_update_t *pUpd;

	switch (ev)
	{
	case HS_DHCP_EV_CFG_UPDATE:
		pUpd = (hs_dhcp_update_t *)arg1;
		if (!pUpd) break;

		hssk.local_ip = pUpd->local_ip;
		hssk.local_ip_valid = 1;

		sPar.local_ip = hssk.local_ip;
```

```
                sPar.local_ip_valid = hssk.local_ip_valid;
                sPar.gateway_ip = pUpd->router_ip;
                sPar.mask = pUpd->subnet_mask;
                HsIpSetParams(&sPar);

                sTcpPar.local_ip = hssk.local_ip;
                sUdpPar.local_ip = hssk.local_ip;
                sArpPar.local_ip = hssk.local_ip;
                sDnsPar.server_ip = pUpd->dns_ip;

                HsUdpSetParams(&sUdpPar);
                HsTcpSetParams(&sTcpPar);
                HsArpSetParams(&sArpPar);
                HsDnsSetParams(&sDnsPar);

                HsSockInetNtoa(hssk.local_ip, ipaddrstr);

                sprintf(str, "Local IP address acquired: %s", ipaddrstr);
                hssk.log_fn(str);

                hssk.global_ev_fn(0, HS_SOCK_EV_DHCP_UPDATE, (long)pUpd, 0, 0, 0);
                break;

        case HS_DHCP_EV_TIMEOUT:
        case HS_DHCP_EV_ERROR:
                hssk.local_ip_valid = 0;
                hssk.local_ip = 0;

                sPar.local_ip = hssk.local_ip;
                sPar.local_ip_valid = hssk.local_ip_valid;
                HsIpSetParams(&sPar);

                sTcpPar.local_ip = hssk.local_ip;
                sUdpPar.local_ip = hssk.local_ip;
                sArpPar.local_ip = hssk.local_ip;

                HsUdpSetParams(&sUdpPar);
                HsTcpSetParams(&sTcpPar);
                HsArpSetParams(&sArpPar);

                hssk.log_fn("Lost of failed to acquire local IP address");
                break;
        }
}
```

2.2.6.2.4 HsDnsGetIpbyName

Declaration:

extern int HsDnsGetIpbyName(unsigned char *name, unsigned long *padre);

Summary:

Various application level protocol libraries of HsTCPIPv4 use this function to resolve DNS name to the IP address. It can also be used from any user application. HsDns will first check internal cache to try to find IP address corresponding to requested name. If name entry is found in internal cache, HsDnsGetIpbyName returns

immediately with success. If entry not found in cache, HsDns will contact DNS server. In this case, HsDnsGetIpbyName returns HS_DNS_RC_PENDING. The caller must allow sufficient time for HsDns to receive reply from DNS server and then call HsDnsGetIpbyName again (This can be achieved using a timer, for example 500 ms).

On second call to HsDnsGetIpbyName, HsDns would have received DNS server reply for name request and would have put it in internal cache, so if request was successful, HsDnsGetIpbyName would return IP address for name entry immediately from cache.

Parameters:

***name – pointer to null terminated string containing DNS name to resolve**

***ipaddr – pointer to 32 bit unsigned integer to receive IP address**

Return values:

- **HS_DNS_RC_NOT_INIT – Library not initialized**
- **HS_DNS_RC_OK – IP address obtained, copied into *ipaddr**
- **HS_DNS_RC_BAD_PARAMS – invalid parameter(s)**
- **HS_DNS_RC_NO_MEM – no memory for new request**
- **HS_DNS_RC_FAIL – HsDns failed to open UDP session for new request or timer failure**
- **HS_DNS_RC_PENDING – New DNS request has been initiated, call back later to check result (see summary description above)**

Sample usage:

```
dns_rc = HsDnsGetIpbyName(dest_host, &addr);
```

2.2.7 HsDhcp

2.2.7.1 Overview

HsDhcp module implements client side of Dynamic Host Configuration Protocol (DHCP) according to RFC 2131. HsSock module updates all relevant components of HsTCPIPv4 stack with new network parameters received from HsDhcp.

Supported network parameters include

- IP address,
- Router IP address,
- Network mask,
- DNS server IP address.

HsDhcp functions include:

- Requesting network parameters from a DHCP server
- Communicating new IP parameters to user of DHCP (HsSock)
- IP lease management, renewal, re-binding and release

2.2.7.2 HsDhcp API

2.2.7.2.1 HsDhcpInit

Declaration:
extern int HsDhcpInit(hsdhcp_init_t *init);

Summary:
Initializes HsDhcp library. This function must be called before any other functions of HsDhcp are called.

Parameters:
hsdhcp_init_t *init – initialization structure defined as follows:

Data member	Description
unsigned char MacAddr[6];	Local Ethernet MAC address
hs_dhcp_event_fn_t *event_fn	Event callback function pointer, defined as typedef void hs_dhcp_event_fn_t(int ev, long arg1, long arg2); Possible event codes and arguments are described in the following section(s).
unsigned long server_ip	32 bit DNS server ip address. If using static IP address, the user may set this value to DNS IP address, otherwise if using DHCP, set to 0. DNS IP address can be communicated to HsDns at the time when it is obtained from DHCP server. In this case the DNS IP address is set with function HsDnsSetParams.

hs_dhcp_timer_start_t *timer_start	Timer start callback function pointer. HsDhcp will use this function when it needs to start a timer
	
	Prototype:
	typedef long hs_dhcp_timer_start_t(unsigned long timeout_ms, void *arg, hs_dhcp_timer_cb_t *cb);
	
	timeout_ms – timeout in milliseconds
	arg – argument passed from HsDhcp
	
	cb – User code must call this function pointer callback on timer expiry
	
	Timer expiry function prototype:
	typedef void hs_dhcp_timer_cb_t(void *arg);
	
	The timer_start function returns long timer handle
	
	arg – same argument as passed to timer_start function

hs_dhcp_timer_stop_t *timer_stop	Timer start callback function pointer. HsDhcp will use this function when it needs to stop a timer Prototype: typedef void hs_dhcp_timer_stop_t(long timer_han); timer_han – timer handle obtained with call to timer_start.
hs_dhcp_get_ticks_fn_t *get_rand32	Pointer to callback function used by HsDhcp when it needs to generate a 32 bit random number. Prototype: typedef unsigned long hs_dhcp_get_ticks_fn_t(void); Returns: 32 bit pseudo random number.
hs_dhcp_log_fn_t *log_fn	Pointer to callback function used by HsDhcp to log debug events. Prototype: typedef void hs_dhcp_log_fn_t(char *str); str – string to be printed or put into event log

Return values:

- **HS_DHCP_RC_ALR_INIT – Library already initialized**
- **HS_DHCP_RC_BAD_PARAM – Invalid parameter(s)**
- **HS_DHCP_RC_OK – HsDhcp initialized successfully**

Sample usage:

```
hssk.dhcp_client = init->dhcp_client;
if (init->dhcp_client)
{
        hssk.local_ip_valid = 0;

        dhcp_init.event_fn = hssock_dhcp_event_callback;
        dhcp_init.get_rand32 = init->get_rand32;
        memcpy(dhcp_init.MacAddr, init->local_eth, 6);
        dhcp_init.timer_start = init->timer_start_fn;
        dhcp_init.timer_stop = init->timer_stop_fn;
        dhcp_init.log_fn = init->log_fn;

        rc = HsDhcpInit(&dhcp_init);
        if (rc != HS_DHCP_RC_OK)
                return HS_SOCK_RC_FAIL;
}
```

2.2.7.2.2 HsDhcpCleanUp

Declaration:
extern void HsDhcpCleanUp(void);

Summary:
De-allocates resources and closes HsDhcp library. Currently this function is called from HsSock library when user calls HsSockCleanUp();

Parameters:
None

Return values:
None

Sample usage:

HsDhcpCleanUp();

2.2.7.2.3 HsDhcpRenew

Declaration:
extern int HsDhcpRenew(void);

Summary:
Initiates IP address renew procedure. HsDhcp requests IP address lease from DHCP server and obtains IP parameters. When IP lease is obtained it calls event callback function and communicates asynchronously new IP parameters to caller.

If the use of DHCP is enabled, HsDhcpRenew is called by HsSock library at the end of HsSockInit function when the initialization of various components of HsTCPIPv4 stack is otherwise complete.

Parameters:
None

Return values:

- **HS_DHCP_RC_NOT_INIT – Library not initialized**
- **HS_DHCP_RC_OK – IP renewal initiated. The result shall be notified to caller asynchronously via event callback.**

Sample usage:

rc = HsDhcpRenew();

2.2.7.2.4 HsDhcpRelease

Declaration:
extern int HsDhcpRelease(void);

Summary:
Releases IP address used by HsTCPIPv4 stack.

Parameters:
None

Return values:

- **HS_DHCP_RC_NOT_INIT – Library not initialized**
- **HS_DHCP_RC_OK – IP address released. This IP address must not be used by HsTCPIPv4 stack from this point on.**

Sample usage:

HsDhcpRelease();

2.2.7.3 HsDhcp Events passed to event callback

Event	Description

HS_DHCP_EV_CFG_UPDATE	IP parameters update. Happens in 2 cases:
	1) New IP lease obtained from DHCP server
	2) IP lease expired and failed to renew lease
	
	Arg1 is a pointer to structure hs_dhcp_update_t defined as:
	
	unsigned long local_ip; // local IP address
	unsigned long dns_ip; // dns server ip address
	unsigned long router_ip; // gateway ip address
	unsigned long subnet_mask; // subnet mask
	unsigned long lease_time; // lease time in seconds
	unsigned long renew_time; // renew time in seconds
	unsigned long rebind_time; // rebind time in seconds
	
	Arg2 = 0

HS_DHCP_EV_TIMEOUT	Dhcp operation timed out, failed to obtain IP parameters Arg1 = 0 Arg2 = 0

HS_DHCP_EV_TIMEOUT	Internal error, this could indicate error trying to open UDP socket or DHCP server rejected DHCP request with NAK Arg1 = 0 Arg2 = 0

2.3 Session Layer API

2.3.1 HsSock

2.3.1.1 Overview

HsSock module provides a common API interface to network level protocols fusing UDP and TCP protocols. The services of HsSock are used by HsSmtp, HsPop3, HsFtp, HsTftp, HsNtp, HsDns, HsDhcp. HsSock interface can also be used directly from a user application. The function of HsSock module is session management

2.3.1.2 HsSock API

2.3.1.2.1 HsSockInit

Declaration:
extern int HsSockInit(hssock_init_t *init);

Summary:
This function initializes HsSock library and must be called first, before any other functions of HsSock are called. Internally HsSockInit will initialize various components of HsTCPIPv4: HsUdp, HsTcp, HsIp, HsDhcp, HsDns.

Parameters:
hssock_init_t *init – initialization structure defined as follows:

Variable	Description
unsigned char *local_eth;	Pointer to 6 byte buffer representing local Ethernet MAC address
int dhcp_client	0=use static IP configuration, 1=Use DHCP client to obtain IP parameters
unsigned char *source_ip_str;	local interface IP address string in dotted format, zero terminated (e.g “192.168.1.1”) Set to “0.0.0.0” if using DHCP.
unsigned char *gateway_ip_str;	Pointer to Gateway router IP address string in dotted format, zero terminated (e.g “192.168.1.9”) Set to “0.0.0.0” if using DHCP.
unsigned char *network_mask;	Pointer to Local network mask string in dotted format, zero terminated (e.g “255.255.255.0”) Set to “0.0.0.0” if using DHCP.
unsigned char *dns_server_ip_str;	Pointer to DNS server IP address in dotted format, zero terminated (e.g “192.168.1.9”) Set to “0.0.0.0” if using DHCP.

hs_sock_event_fn_t *global_ev_fn;	**Pointer to global socket event function callback.** **This function shall be called by HsSock when it needs to communicate events to application that are not related to specific socket session, such as DHCP updates.** **Prototype:** **typedef long hs_sock_event_fn_t(long user_handle, int ev, long arg1, long arg2, long arg3, long arg4);** **Possible Event code and arguments are specified in HsSock Events section.**
hs_sock_timer_start_t *timer_start_fn	**Start timer function callback. HsSock calls this function pointer when it needs to start a timer.** **Prototype:** **typedef long hs_sock_timer_start_t(unsigned long timeout_ms, void *arg, hs_sock_timer_cb_t *cb);** **timeout_ms – timeout in milliseconds** **arg – argument passed by HsSock** **cb – callback function to be called when timer expires.** **Returns timer handle** **Callback function prototype for timer expiry:** **typedef void hs_sock_timer_cb_t(void *arg);** **arg – same argument as passed to timer_start_fn**
hs_sock_timer_stop_t *timer_stop_fn;	**Stop timer function callback HsSock calls this function pointer when it needs to stop a timer.** **Prototype:** **typedef void hs_sock_timer_stop_t(long timer_han);** **timer_han – timer handle**
hs_sock_drv_tx_fn_t *drv_tx_fn;	**Ethernet frame transmit callback function pointer.**

	HsSock calls this function when it needs to transmit an Ethernet frame. **Prototype:** **typedef void hs_sock_drv_tx_fn_t(unsigned char *pkt, int len);** **pkt – buffer to packet to be transmitted** **len – length of buffer in bytes**
hs_sock_get_ticks_fn_t *get_ticks_fn;	**Pointer to callback function used to get number of millisecond ticks since bootup** **Prototype:** **typedef unsigned long hs_sock_get_ticks_fn_t(void);** **Returns number of millisecond ticks since bootup.**
unsigned long seed;	**32 Bit initial seed to be used for random number generation.**
hsicmp_event_fn_t *icmp_event_fn;	**Pointer to ICMP event callback function. If user application uses ICMP interface in HsIcmp module to ping remote hosts, this function gets called to report ICMP ping related events.** **Prototype:** **typedef long hsicmp_event_fn_t(int ev, long arg1, long arg2);** **ev –event code** **arg1, arg2 – arguments. See section HsSock events for detailed specification.**

Return values:

- **HS_SOCK_RC_OK – Success, HsSock and HsTCPIPv4 stack initialized**
- **HS_SOCK_RC_PARAM – Invalid parameters**
- **HS_SOCK_RC_FAIL – Initialization failed**
- **HS_SOCK_RC_SOCKINIT_FAILED - Initialization failed**

Sample usage:

```
sockinit.dhcp_client = (IsDlgButtonChecked(hDlg, IDC_CHECK_DHCP) == BST_CHECKED) ? 1: 0;

if (sockinit.dhcp_client)
        strcpy(source_ipaddr, "0.0.0.0");

sockinit.source_ip_str = source_ipaddr;
```

```
sockinit.gateway_ip_str = gw_ipaddr;
sockinit.dns_server_ip_str = dns_ipaddr;
sockinit.network_mask = mask;
sockinit.local_eth = ethaddr;
sockinit.timer_start_fn = sock_timer_start;
sockinit.timer_stop_fn = sock_timer_stop;
sockinit.drv_tx_fn = sock_drv_tx_fn;
sockinit.log_fn = write_event;
sockinit.global_ev_fn = hs_sock_global_event;

sockinit.get_ticks_fn = hs_sock_get_ticks;
sockinit.seed = (unsigned long)GetTickCount();
sockinit.get_rand32 = hs_sock_get_rand32;

sockinit.icmp_event_fn = icmp_event_callback;

rc = HsSockInit(&sockinit);
if (rc != HS_SOCK_RC_OK)
{
        sprintf(str, "HsSockInit failed, rc=%s", HsSockGetRcString(rc));
        write_event(str);
        break;
}

write_event("HsTCPIPv4 Initialized");
```

2.3.1.2.2 HsSockCleanUp

Declaration:
HsSockCleanUp(void);

Summary:
De-initializes HsSock module and HsTCPIPv4 stack.

Parameters:
None

Return values:
None

Sample usage:

HsSockCleanUp();

2.3.1.2.3 HsSockUdpOpen

Declaration:

```
extern int HsSockUdpOpen(
        long                usr_handle,    // user supplied handle
        hs_sock_event_fn_t  *event_fn,     // event callback function
        unsigned short      source_port,   // source port
        long                *sock_handle);// returned socket layer handle;
```

Summary:
Opens UDP socket session. This call enables an application to both start receiving UDP data sent from remote hosts to port source_port and to send UDP data to remote hosts from port source_port

Parameters:

usr_handle – application handle to associate with new socket session
event_fn – event callback function to be called by HsSock with socket related events.

Callback function prototype:
typedef long hs_sock_event_fn_t(long user_handle, int ev, long arg1, long arg2, long arg3, long arg4);

See HsSock Events for detailed specification.

source_port – local port number. To have HsSock allocate local port number dynamically, set to 0.

***sock_handle – pointer to variable of type long to receive UDP socket session handle**

Return values:

- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**
- **HS_SOCK_RC_BIND_FAILED – A socket session with specified port number already exists, failed to create new session.**
- **HS_SOCK_RC_NO_MEM – No memory for new socket session. Maximum number of concurrent sessions reached. Currently HsSock supports maximum 11 concurrent sessions.**
- **HS_SOCK_RC_OK - Success**

Sample usage:

The following code is an example how HsSockUdpOpen is used by HsDns:

```
// Initialize state
static void hs_dhcp_st_init(int ev, long arg1, long arg2, long arg3)
{
	int rc;
	unsigned long timeout_ms;

	switch (ev)
	{
	case HS_DHCP_FSM_EV_DISCOVER:
		dhcp.xid = dhcp.api.get_rand32();

		dhcp.local_ip = 0;
		dhcp.server_ip = 0;
		dhcp.tmp_my_ip = 0;
		dhcp.tmp_server_ip = 0;

		rc = HsSockUdpOpen((long)dhcp.xid, hs_udp_sock_event, HS_DHCP_CPORT, &dhcp.sock_han);

		if (rc != HS_SOCK_RC_OK)
		{
			dhcp.api.event_fn(HS_DHCP_EV_ERROR, 0, 0);
			break;
		}
```

```
                hs_dhcp_send_discover();

                hs_dhcp_change_state(hs_dhcp_st_select);

                dhcp.retry_secs = 4;
                timeout_ms = (unsigned long)(get_random_between(
                        (unsigned short)(dhcp.retry_secs-1),
                        (unsigned short)(dhcp.retry_secs+1)) * 1000);

                hs_dhcp_start_timer(&dhcp.retx_timer, timeout_ms, HS_DHCP_FSM_EV_T0);
                break;

        default:
                break;
        }
}
```

2.3.1.2.4 HsSockTcpConnect

Declaration:

```
int HsSockTcpConnect(
        long                    usr_handle,         // user handle
        hs_sock_event_fn_t      *event_fn,          // socket event callback
        unsigned char           *dest_ip,           // remote ip address to connect to
        unsigned short          dest_port,          // remote port
        long                    *sock_handle);      // socket handle returned;
```

Summary:

This function is used to initiate outgoing TCP connection to remote host.

This function is non-blocking and returns immediately after validation of parameters and internal conditions. It initiates TCP connection. The actual result of operation is reported asynchronously via event callback (*event_fn). See event specification in section HsSock Events.

Parameters:

usr_handle – user handle to associate with new socket session

***event_fn – socket event callback. Defined as:**
typedef long hs_sock_event_fn_t(long user_handle, int ev, long arg1, long arg2, long arg3, long arg4); See detailed description of events in section HsSock Events.

***dest_ip – null terminated string representing remote IP address to connect to in dotted IP format (e.g. “192.168.1.3”)**

dest_port – remote port to connect to

***sock_handle – long variable that receives socket session handle**

Return values:

- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**

- **HS_SOCK_RC_FAIL – Internal error : HsTcpConnect failed immediately.**
- **HS_SOCK_RC_NO_MEM – No memory for new socket session. Maximum number of concurrent sessions reached. Currently HsSock supports maximum 11 concurrent sessions.**
- **HS_SOCK_RC_OK – Success, connection initiated**

Sample usage:

```
rc = HsSockTcpConnect((long)pSession, sock_event_fn_tcp, pSession->remote_ip_str,
        pSession->remote_port, &pSession->sock_han);

if (rc != HS_SOCK_RC_OK)
{
        sprintf(str, "HsSockTcpConnect failed rc=%s", HsSockGetRcString(rc));
        write_event(str);
        free_session(pSession);
        return;
}

LvAddSession(pSession);

write_event("HsSockTcpConnect initiated OK.");
```

2.3.1.2.5 HsSockTcpListen

Declaration:

```
int HsSockTcpListen(
        unsigned short      port,               // port to listen on
        long                user_handle,        // user handle
        hs_sock_event_fn_t  *event_fn,          // socket event callback
        long                *listen_sock_han);  // listen socket handle returned;
```

Summary:

This function is used to start listening for incoming TCP connections on specified local port.

Parameters:

port – local port number to listen for incoming TCP connections on
user_handle – user handle to associate with a listening socket

***event_fn - socket event callback. Defined as:**
typedef long hs_sock_event_fn_t(long user_handle, int ev, long arg1, long arg2, long arg3, long arg4); See detailed description of events in section HsSock Events.

***listen_sock_han – listening socket handle returned**

Return values:

- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**
- **HS_SOCK_RC_FAIL – Internal error : HsTcpListen failed immediately.**

- **HS_SOCK_RC_NO_MEM – No memory for listening socket session. Maximum number of concurrent sessions reached. Currently HsSock supports maximum 11 concurrent sessions.**
- **HS_SOCK_RC_OK – Success, started listening for incoming connections**

Sample usage:

```
rc = HsSockTcpListen(local_port, (long)local_port, sock_event_fn_tcp, &listen_sock_han);
if (rc != HS_SOCK_RC_OK)
{
        sprintf(str, "HsSockTcpListen failed rc=%s", HsSockGetRcString(rc));
        write_event(str);
        return;
}

sprintf(str, "HsSockTcpListen OK. Listening on local port %u", local_port);
write_event(str);
```

2.3.1.2.6 HsSockClose

Declaration:
extern int HsSockClose(long sock_handle);

Summary:
This function is used to close an existing socket session.

Parameters:
sock_handle – socket session handle

Return values:
- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**
- **HS_SOCK_RC_OK – Success, session closed**

Sample usage:

```
HsSockClose(handle);
```

2.3.1.2.7 HsSockUdpSendto

Declaration:

```
int HsSockUdpSendto(
        long                hssk_handle,    // socket layer handle
        unsigned long       dest_ip,        // remote end IP address
        unsigned short int  dest_port,      // remote UDP port number
        unsigned char       *buffer,        // payload data
        unsigned short int  length);        // payload data length
```

Summary:
This function is used to send data using UDP protocol on UDP socket session.

Parameters:
hssk_handle – socket session handle obtained after call to HsSockUdpOpen.

dest_ip – 32 bit remote IP address to send data to

dest_port – destination port to send data to

buffer – data buffer to send

length – length of data buffer in bytes

Return values:
- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**
- **HS_SOCK_RC_FAIL – HsUdpSendPacket failed immediately.**
- **HS_SOCK_RC_OK – Success, Udp packet containing data has been sent to HsIp module for transmission**

Sample usage:

```
HsSockUdpSendto(pCtx->sock_handle, pCtx->dest_ip, pCtx->dcid, pCtx->pPkt, pCtx->pktlen);
```

2.3.1.2.8 HsSockTcpSend

Declaration:
```
int HsSockTcpSend(
        long            hssk_handle,  // socket layer handle
        unsigned char   *packet_buf,  // packet data
        int             length,       // packet data length
        int             *txed_len);   // bytes actually transmitted;
```

Summary:
This function is used to send data over established TCP socket session.

Parameters:
hssk_handle – socket handle

packet_buf – data buffer to send

length – length of data buffer to send in bytes

***txed_len – pointer to integer variable that receives count of bytes actually sent.**

Return values:
- **HS_SOCK_RC_NOT_INIT – Library not initialized**
- **HS_SOCK_RC_PARAM – Invalid parameter(s)**
- **HS_SOCK_RC_FAIL – Internal error occurred**
- **HS_SOCK_RC_OK – Success, data has been buffered by TCP for transmission.**
- **HS_SOCK_RC_SEND_PENDING – TCP session transmit queue is full. User must attempt sending data later when TCP is able to accept more data for transmission.**

Sample usage:

```
rc = HsSockTcpSend(pSession->sock_han, buffer, len, &txed_len);

sprintf(str, "HsSockTcpSend rc=%s", HsSockGetRcString(rc));
write_event(str);
```

2.3.1.2.9 HsSockInetAddr

Declaration:
extern int HsSockInetAddr(unsigned char *cp, unsigned long *addr);

Summary:
This function converts an IP address string in dotted IP format to 32 bit IP address

Parameters:
***cp – IP address string**
***addr – pointer to 32 bit integer to receive IP address**

Return values:
1 = conversion successful
0 = conversion failed, the string is not a valid IP address string.

Sample usage:

```
if (!HsSockInetAddr(ntp.s.srv_ip, &addr))
{
        // error
        return;
}
```

2.3.1.2.10 HsSockInetNtoa

Declaration:
extern void HsSockInetNtoa(unsigned long ipl, unsigned char *ip_addr);

Summary:
This function converts 32 bit unsigned integer IP address into zero terminated IP address string in dotted IP format. This function shall terminate the resulting string with zero byte.

Parameters:
ipl – 32 bit unsigned integer ip address

***ip_addr – pointer to buffer to receive IP address string. The buffer must be at least 16 bytes long.**

Return values:
None

Sample usage:

unsigned char str[16];

HsSockInetNtoa(p->local_ip, str);

2.3.1.2.11 HsSockGetRcString

Declaration:
extern unsigned char *HsSockGetRcString(int rc);

Summary:
This function returns a descriptive string corresponding to HsSock return code

Parameters:
rc – HsSock return code

Return values:
A string describing error code

Sample usage:

```
rc = HsSockInit(&sockinit);
if (rc != HS_SOCK_RC_OK)
{
        sprintf(str, "HsSockInit failed, rc=%s", HsSockGetRcString(rc));
        write_event(str);
        return;
}
```

2.3.1.3 HsSock Events

Event	Description
HS_SOCK_EV_DATA	Data received on a socket session. UDP sockets: Arg1 = pointer to data buffer received. Arg2 = length of data received in bytes Arg3 = Remote port number buffer came from Arg4 = Remote IP address (32 bit) data buffer came from HsUdp uses it own internal buffer to receive data into and it passes pointer to this buffer in Arg1. User application must NOT free this buffer, it should copy data into a different buffer or process buffer data and return TCP sockets: Arg1 = pointer to data buffer received Arg2 = length of data received in bytes

	Arg3 = 0 Arg4 = 0 HsSock uses internal buffer to receive data from TCP. User application must NOT free this buffer, it should copy data into a different buffer or process buffer data and return
HS_SOCK_EV_CLOSED	Socket session closed by network User application should consider socket session closed. After callback function returns HsSock releases socket session context. Application must not send any data from this point to this socket Application must not close this socket as it has already been internally closed by HsSock Arg1, Arg2, Arg3, Arg4 all zero
HS_SOCK_EV_CONNECTED	TCP outgoing socket connected Arg1, Arg2, Arg3, Arg4 all zero
HS_SOCK_EV_ACCEPTED	TCP incoming connection has come in from remote host. The user application can instruct HsSock to accept or reject it. <u>To accept new session:</u> Return a non zero user application handle. <u>To reject new session:</u> Return 0 Arguments: Arg1: listening socket user handle, same as passed to HsSockTcpListen Arg2: remote port TCP connection came from Arg3: remote IP address TCP connection came from (32 bit integer) Arg4: local port TCP connection came to
HS_SOCK_EV_CONN_FAILED	TCP connect attempt failed Arg1, Arg2, Arg3, Arg4 all zero
HS_SOCK_EV_DHCP_UPDATE	IP parameters acquired / changed by DHCP module

	HsSock has already configured all relevant HsTCPIPv4 components with new IP parameters. The application may also use the data in this event to update for example to update IP address in the GUI or print event showing has new IP address has been acquired.
	
	Arg1 points to structure hs_dhcp_update_t defined as follows:
	
	// update event data
	typedef struct
	{
	unsigned long local_ip; // local IP address
	unsigned long dns_ip; // dns server ip address
	unsigned long router_ip; // gateway ip address
	unsigned long subnet_mask; // subnet mask
	unsigned long lease_time; // lease time in seconds
	unsigned long renew_time; // renew time in seconds
	unsigned long rebind_time; // rebind time in seconds
	} hs_dhcp_update_t;
	
	Arg2, Arg3, Arg4 all zero

2.3.1.4 ICMP Event Callback Events

Event	Description
HS_ICMP_EV_PING_TIMEOUT	ICMP echo request timed out Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_SENT	ICMP echo request sent Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_REPLY	ICMP echo reply received Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_FINISHED	HsIcmp finished ping sequence: Arg1= number of ping requests sent Arg2 = number of ping replies received

2.4 Transport Layer API

2.4.1 HsTcp

2.4.1.1 Overview

HsTcp module implements TCP (Transport Control Protocol) as per RFC 793. It is the most complex part of HsTCPIPv4 suite and is responsible for many functions, such as:

- **Incoming and outgoing connection establishment / connection management, selection of unique initial sequence numbers,**
- **TCP options negotiation**
- **Reliable data transmission, error detection, re-transmissions**
- **Flow control, congestion control**

2.4.1.2 HsTcp API

2.4.1.2.1 HsTcpInit

Declaration:

extern int HsTcpInit(hstcp_init_t *init);

Summary:

This function initializes HsTcp library. It must be called once before any other functions of HsTcp are called

Parameters:

hstcp_init_t *init – initialization structure defined as follows:

Variable	Description
unsigned long local_ip;	Local IP address. Set to 0 is DHCP is used
int max_sessions;	Maximum number of concurrent TCP sessions required to support
unsigned long max_retrans_timeout;	Maximum retransmission timeout value in milliseconds
unsigned long min_retrans_timeout;	Minimum retransmission timeout value in milliseconds
unsigned short data_tx_attempts;	Maximum number of data retransmissions
unsigned long connect_timeout;	TCP SYN retransmission timeout in milliseconds
unsigned long wait_state_timeout;	Time to wait in wait state before releasing TCP context (in milliseconds
hs_tcp_get_ticks_fn_t *get_ticks_fn;	Pointer to function callback to get number of millisecond ticks since bootup Prototype: typedef unsigned long hs_tcp_get_ticks_fn_t(void); Returns number of milliseconds since bootup.
unsigned long seed;	32 bit seed value to be used for random number generation
hs_tcp_timer_start_t *start_timer;	Pointer to function callback to start a timer

	HsTcp calls this function whenever it needs to start a timer.
	
	Prototype:
	
	typedef long hs_tcp_timer_start_t(unsigned long timeout_ms, void *arg, hs_tcp_timer_cb_t *cb);
	
	timeout_ms – timeout in milliseconds
	
	arg – argument passed in by HsTcp
	
	cb – timer expiry callback, define as:
	typedef void hs_tcp_timer_cb_t(void *arg);
	
	arg – argument same as passed to start_timer
	
	Returns timer handle

hs_tcp_timer_stop_t *stop_timer;	Pointer to function callback to stop a timer HsTcp calls this function to stop a timer. Prototype: typedef void hs_tcp_timer_stop_t(long timer_han); timer_han – timer handle as returned by start_timer.
hs_tcp_get_buf_t *get_buffer;	Pointer to function callback to allocate a buffer for reception of TCP segment data HsTcp calls this function whenever it needs a data buffer to copy incoming data arrived on TCP connection into. Prototype: typedef unsigned char *hs_tcp_get_buf_t(long user_handle, unsigned short int length); user_handle – user handle associated with TCP session. Length – number of bytes required in buffer Returns buffer pointer or NULL. HsTcp does not free this buffer. Caller may free this buffer after data has been delivered to user via event callback.
hs_tcp_log_fn_t *log_fn;	Pointer to function callback to log debug events Prototype: typedef void hs_tcp_log_fn_t(char *str); str – string to print or put into event log
unsigned short int txq_size;	Size of transmit queue for data accepted from user of TCP for transmission in bytes

Return values:

- **HS_TCP_RC_ALR_INIT – Library already initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_NO_MEM – not enough memory**
- **HS_TCP_RC_OK - success**

Sample usage:

```
tcp_init.local_ip = hssk.local_ip;
tcp_init.max_sessions = HSSOCK_MAX;
tcp_init.get_ticks_fn = init->get_ticks_fn;
tcp_init.seed = init->seed;
tcp_init.start_timer = init->timer_start_fn;
tcp_init.stop_timer = init->timer_stop_fn;
tcp_init.get_buffer = tcp_get_buf;
tcp_init.log_fn = init->log_fn;
tcp_init.txq_size = 4096;

rc = HsTcpInit(&tcp_init);
```

2.4.1.2.2 HsTcpSetParams

Declaration:
extern void HsTcpSetParams(hs_tcp_params_t *pParams);

Summary:
This function sets TCP/IP parameters at run time. Currently used to update TCP with a new local IP address after it has been obtained from DHCP server.

Parameters:
hs_tcp_params_t *pParams – pointer to parameter structure, defined as:

Variable	Description
unsigned long local_ip;	Local IP address

Return values:
None

Sample usage:

```
hs_tcp_params_t sPar = {0};

sPar.local_ip = new_ip;

HsTcpSetParams(&sPar);
```

2.4.1.2.3 HsTcpCleanUp

Declaration:
extern void HsTcpCleanUp(void);

Summary:
This function is used to de-initialize HsTcp library.

Parameters:
None

Return values:
None

Sample usage:

HsTcpCleanUp();

2.4.1.2.4 HsTcpConnect

Declaration:

```
extern
int HsTcpConnect(
        long    usr_handle,             // user handle
        unsigned long  dest_ip,         // destination IP address to connect to
        unsigned short source_port, // source port
        unsigned short dest_port,   // destination port
        hs_tcp_event_fn_t *event_callback, // event callback
        long *tcp_handle);              // TCP new session handle;
```

Summary:
This function is used to initiate outgoing TCP connection to remote host. The result of connection establishment operation is reported asynchronously via event callback. See section HsTcp Events

Parameters:
usr_handle – user handle to associate with TCP connection.

dest_ip – destination IP address

source_port – local port

dest_port - destination port

***event_callback – pointer to event callback function to be called to communicate network events for this connection**

***tcp_handle – pointer to variable of type long to receive TCP connection handle.**

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_NO_MEM – not enough memory**
- **HS_TCP_RC_OK - success**

Sample usage:

```
rc = HsTcpConnect((long)skp, addr, skp->source_port, dest_port, tcp_event_handler,
```

&skp->conn_handle);

2.4.1.2.5 HsTcpListen

Declaration:

extern
int HsTcpListen(long usr_handle, unsigned short port, unsigned long remote_ip,
hs_tcp_event_fn_t *event_callback, long *listen_handle);

Summary:
This function is used to start listening for a new TCP connection on specified local port from specified remote IP address. When an incoming connection is received matching specified parameters, user is notified via event callback. See HsTcp Events section.

Parameters:
usr_handle – user handle to associate with TCP connection.

port – local port to listen on

remote_ip – remote IP address to accept TCP connection from. Set to 0 to receive connection requests from all hosts.

***event_callback – TCP calls this event callback function to report network events associated with new connection**

***listen_handle – pointer to variable to receive listen handle.**

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_NO_CTX – reached maximum number of TCP sessions, cannot create new session**
- **HS_TCP_RC_OK - success**

Sample usage:

rc = HsTcpListen((long)skp, port, 0, tcp_event_handler, &skp->listen_handle);

2.4.1.2.6 HsTcpBindSession

Declaration:
int HsTcpBindSession(long conn_handle, long user_han);

Summary:
When a new incoming connection request arrives from remote host this function is used to accept incoming connection and bind a user handle with a new TCP session.

Parameters:
conn_handle – TCP session handle
user_han – user handle to associate with TCP session

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_OK - success**

Sample usage:

HsTcpBindSession(tcp_context, (long)skp);

2.4.1.2.7 HsTcpStopListen

Declaration:
extern int HsTcpStopListen(long listen_handle);

Summary:
This function is called to stop listening for incoming connections.

Parameters:
listen_handle – listen handle obtained with a call to HsTcpListen

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_OK - success**

2.4.1.2.8 HsTcpClose

Declaration:
extern int HsTcpClose(long tcp_handle);

Summary:
This function is used to close a TCP session.

Parameters:
tcp_handle – session handle to close

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_OK - success**

Sample usage:

HsTcpClose(handle);

2.4.1.2.9 HsTcpSend

Declaration:

```
extern
int HsTcpSend(long   conn_handle,        // TCP connection handle
              unsigned char *buffer,     // packet data
              int length,                // packet data length
              int *txed_len);            // bytes actually transmitted
```

Summary:

This function is used to send data over TCP session.

Parameters:

conn_handle – TCP session handle

buffer – data buffer to send

length – length of buffer to send in bytes

***txed_len – variable to receive count of bytes actually sent**

Return values:

- **HS_TCP_RC_NOT_INIT – Library not initialized**
- **HS_TCP_RC_BAD_PARAM – Invalid parameters**
- **HS_TCP_RC_INV_STATE – cannot queue data for transmission because connection state is not established**
- **HS_TCP_RC_TXQ_FULL – TCP session input transmit queue is full. User must call later when TCP is able to accept more data from user for transmission**
- **HS_TCP_RC_OK - success**

Sample usage:

```
rc = HsTcpSend(skp->conn_handle, packet_buf, length, txed_len);
```

2.4.1.2.10 HsTcpReceivePacket

Declaration:

```
extern void HsTcpReceivePacket(unsigned char *pTcpPkt, unsigned short tcp_len);
```

Summary:

This function is called to pass a received TCP packet to from IP layer (HsIp).

Parameters:

pTcpPkt – pointer to start of TCP packet

tcp_len – length of TCP packet

Return values:

None

Sample usage:

```
HsTcpReceivePacket(pIpPkt, total_len);
```

2.4.1.3 HsTcp Events

Event	Description
HS_TCP_USR_EV_CONNECT	Incoming TCP connection request received matching previously submitted listen with the HsTcpListen function. The user must accept or reject TCP connection: To accept, call: HsTcpBindSession To reject, call: HsTcpClose Arguments passed with HS_TCP_USR_EV_CONNECT: Arg1 = TCP session handle of new session Arg2 = remote port Arg3 = remote IP address
HS_TCP_USR_EV_DATA	Data received from TCP session: Arg1 = pointer to buffer containing data Arg2 = length of data in bytes Arg3 = 0
HS_TCP_USR_EV_RELEASE	TCP session closed Arg1 = 0; Arg2 = 0; Arg3 = 0;
HS_TCP_USR_EV_ACCEPT	Outgoing TCP session connected Arg1 = 0; Arg2 = 0; Arg3 = 0;

2.4.2 HsUdp

2.4.2.1 Overview

HsUdp module implements UDP (User Datagram Protocol) as per RFC 768.
HsUdp functions include:

- **Building and sending UDP datagrams using HsIP module**
- **Processing UDP datagrams received from HsIP module**

2.4.2.2 HsUdp API

2.4.2.2.1 HsUdpInit

Declaration:
extern void HsUdpInit(unsigned long local_ip, hs_sock_rxfn_t *RxFn, log_fn_t *logfn);

Summary:
This function initializes HsUdp library. This function must be called first before any other functions of HsUdp are called

Parameters:

unsigned long local_ip – local IP address

hs_sock_rxfn_t *RxFn – pointer to callback function HsUdp is to call when it has received UDP data.

Prototype:

```
typedef void hs_sock_rxfn_t
(
        unsigned long       remote_ip,      // remote ip address
        unsigned short int  local_port,     // local port
        unsigned short int  remote_port,    // remote port
        int                 protocol,       // protocol
        unsigned char       *RxBuf,         // received buffer
        unsigned short int  rxlen           // received length
);
```

log_fn_t *logfn – event log function callback. HsUdp can call this function to log debug events

Return values:
None

Sample usage:

```
HsUdpInit(hssk.local_ip, hs_sock_receive_data, init->log_fn);
```

2.4.2.2.2 HsUdpSetParams

Declaration:
extern void HsUdpSetParams(hs_udp_params_t *pParams);

Summary:
This function is used to update HsUdp module with new parameters. Currently used to update local IP address when it is received from DHCP server (if DHCP protocol and not static IP is used).

Parameters:
hs_udp_params_t *pParams – parameter structure as follows:

```
// UDP parameters
typedef struct
{
        unsigned long local_ip;                         // local ip address
} hs_udp_params_t;
```

Return values:
None

Sample usage:

```
HsUdpSetParams(&sUdpPar);
```

2.4.2.2.3 HsUdpCleanUp

Declaration:
extern void HsUdpCleanUp(void);

Summary:
This function is used to de-initialize HsUdp module.

Parameters:
None

Return values:
None

Sample usage:

```
HsUdpCleanUp();
```

2.4.2.2.4 HsUdpSendPacket

Declaration:

```
int HsUdpSendPacket(
        unsigned long       dest_ip,        // remote end IP address
        unsigned short int  dest_port,      // remote UDP port number
        unsigned short int  source_port,    // source UDP port number
        unsigned char       *payload_buf,   // UDP payload to send
        unsigned short int  payload_len);   // UDP payload length
```

Summary:

This function is called to send UDP data.

Parameters:

dest_ip – destination IP address
dest_port – destination port
source_port – local port
payload_buf – data buffer to send
payload_len – length of data to send

Return values:

1 = data has been transmitted
0 = data transmission failed

Sample usage:

```
if (!HsUdpSendPacket(dest_ip, dest_port, skp->source_port, pPkt, length))
        return HS_SOCK_RC_FAIL;
```

2.4.2.2.5 HsUdpReceivePacket

Declaration:

```
extern void HsUdpReceivePacket(unsigned char *pUdpPkt, int udplen);
```

Summary:

This function is called to pass received UDP packet to HsUdp module for further processing.

Parameters:

*pUdpPkt – pointer to start of UDP packet
Udplen – length of UDP packet

Return values:

none

Sample usage:

```
/* process reassembled datagram */
static void hsip_process_complete_ip_datagram(
        unsigned char   *pIpPkt,// ip packet
        unsigned short  total_len,      // ip packet data length (not including header)
        unsigned char   protocol,       // protocol
```

```
        unsigned long   src_ip              // source ip address
)
{
        switch (protocol)
        {
        case HS_IP_UDP_PROTO:
                HsUdpReceivePacket(pIpPkt, (int)total_len);
                break;

        case HS_IP_TCP_PROTO:
                HsTcpReceivePacket(pIpPkt, total_len);
                break;

        case HS_IP_ICMP_PROTO:
                HsIcmpReceivePacket(src_ip, pIpPkt, (int)total_len);
                break;

        default:
                return;
        }
}
```

2.5 Network Layer API

2.5.1 HsIp

2.5.1.1 Overview

HsIP module implements IP (Internet Protocol) layer as specified in RFC 791. HsIp primary functions include:

- Building IP packets and transmitting to the network using Ethernet packet driver API
- Resolving destination IP address to Ethernet MAC address using HsArp module API
- IP packet fragmentation if payload exceeds configured MTU
- Processing received IP datagrams
- IP fragment re-assembly

2.5.1.2 HsIp API

2.5.1.2.1 HsIpInit

Declaration:
extern int HsIpInit(hs_ip_init_t *pInit);

Summary:
This function initialized HsIp library. This function must be called before any other functions of HsIp are called.

Parameters:
hs_ip_init_t *pInit pointer to parameter structure defined as follows:

Variable	Description
unsigned long local_ip	**Local IP address**
int local_ip_valid	**1= Local IP address is valid. 0=Address is not known**
unsigned long gateway_ip	Gateway (router) IP address
unsigned long mask	**Network mask**
unsigned char *local_eth_addr	**Pointer to local Ethernet MAC address buffer (6 bytes long)**
int mtu	**IP maximum transmission unit in bytes**
hsip_timer_start_t *timer_start	**Pointer to function callback used by HsIp to start a timer** **Prototype:** **typedef long hsip_timer_start_t(unsigned long timeout_ms, void *arg, hsip_timer_cb_t *cb);** **timeout_ms – timeout in milliseconds** **arg – argument that HsIp passed to callback**

		function **cb – function to call on timer expiry** **Returns timer handle** **Timer expiry callback prototype:** **typedef void hsip_timer_cb_t(void *arg);** **arg – argument same as passed to timer_start function**
hsip_timer_stop_t	*timer_stop	**Pointer to function callback used by HsIp to stop a timer** **Prototype:** **typedef void hsip_timer_stop_t(long timer_han);** **time_han – timer handle**
hsip_drv_tx_fn_t	*drv_tx	**Pointer to function callback used by HsIp to Transmit IP packet** **Prototype:** **typedef void hsip_drv_tx_fn_t(unsigned char *pkt, int len);** **pkt – IP packet buffer** **len – length of IP packet buffer in bytes**
hsip_log_fn_t	*log_fn	**Pointer to function callback used by HsIp to log a debug event** **Prototype:** **typedef void hsip_log_fn_t(char *str);** **str – event string to print or to put intop event log**
hsicmp_event_fn_t	*icmp_event_fn	**ICMP event callback** **Prototype:** **typedef long hsicmp_event_fn_t(int ev, long arg1, long arg2);** **ev – ICMP event code, arguments are event specific. See description of ICMP module**
hsip_get_ticks_fn_t	*get_ticks	**Pointer to function callback used by HsIp to get number of millisecond ticks since bootup** **Prototype:**

	typedef unsigned long hsicmp_get_ticks_fn_t(void); **returns number of millisecond ticks since bootup**

Return values:
1 = Success
0 = Failed

Sample usage:

```
ip_init.local_eth_addr = init->local_eth;
ip_init.local_ip = hssk.local_ip;
ip_init.local_ip_valid = hssk.local_ip_valid;
ip_init.mtu = HS_SOCK_IP_MTU;
ip_init.timer_start = init->timer_start_fn;
ip_init.timer_stop = init->timer_stop_fn;
ip_init.drv_tx = init->drv_tx_fn;
ip_init.log_fn = init->log_fn;
ip_init.get_ticks = init->get_ticks_fn;
ip_init.icmp_event_fn = init->icmp_event_fn;

HsIpInit(&ip_init);
```

2.5.1.2.2 HsIpSetParams

Declaration:
extern void HsIpSetParams(hsip_params_t *params);

Summary:
This function updates HsIp library with new IP parameters when they are received from DHCP server (if DHCP protocol is used).

Parameters:
hsip_params_t *params – pointer to parameters structure defined as

```
typedef struct
{
        unsigned long       local_ip;           // local ip address
        int                 local_ip_valid;     // is local ip address valid
        unsigned long       gateway_ip;         // gateway ip address
        unsigned long       mask;               // local network mask
} hsip_params_t;
```

Return values:
none

Sample usage:

```
sPar.local_ip = hssk.local_ip;
sPar.local_ip_valid = hssk.local_ip_valid;
sPar.gateway_ip = pUpd->router_ip;
sPar.mask = pUpd->subnet_mask;
HsIpSetParams(&sPar);
```

2.5.1.2.3 HsIpShutdown

Declaration:

```
extern void HsIpShutdown(void);
```

Summary:

This function is used to de-initialize HdIp library.

Parameters:

none

Return values:

none

Sample usage:

```
HsIpShutdown();
```

2.5.1.2.4 HsIpSendPacket

Declaration:

```
extern int HsIpSendPacket(unsigned long   dest_ip,   // destination IP address
                          unsigned char   protocol,  // upper layer protocol
                          unsigned char   *pkt,      // IP payload
                          unsigned short  length);   // IP payload length;
```

Summary:

This function is used to send user payload data encapsulated in an IP packet.

Parameters:

dest_ip – destination IP address to send to
protocol – higher layer protocol code
pkt – pointer to IP payload buffer to send
length – length of IP payload buffer in bytes

Return values:

none

Sample usage:

```
HsIpSendPacket(dest_ip, (unsigned char)HS_PROTOCOL_UDP, pUdpPkt, udp_length);
```

2.5.1.2.5 HsIpReceivePacket

Declaration:

```
extern void HsIpReceivePacket(unsigned char *pkt, int len);
```

Summary:
This function is used to pass received IP packet to HsIp for further processing.

Parameters:
pkt – pointer to start of IP packet

Len – length of IP packet in bytes

Return values:
none

Sample usage:

```
while (1)
{
	if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
	{
		if (GetMessage(&msg, NULL, 0, 0))
		{
			if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
			{
				TranslateMessage(&msg);
				DispatchMessage(&msg);
			}
		}
		else
		{
			break;
		}
	}
	else
	{
		Sleep(1);

		if (adhandle)
		{
			r = pcap_next_ex( adhandle, &header, &pkt_data);
			if (r > 0)
			{
				HsIpReceivePacket(pkt_data, (int)header->caplen);
			}
		}

	}
}
```

2.5.1.2.6 HsIpChecksum16

Declaration:
extern unsigned short HsIpChecksum16(unsigned char* data, int len);

Summary:
This function is used to calculate IP checksum on a data buffer.

Parameters:

data – pointer to buffer to calculate checksum on

len – length of buffer in bytes

Return values:

16 bit checksum value

Sample usage:

```
checksum = HsIpChecksum16(pPkt, udplen + HS_UDP_PSHDR_SZ + padding);
```

2.5.2 HsIcmp

2.5.2.1 Overview

HsIcmp module implements ICMP (Internet Control Message Protocol), specifically 2 functions:

- **Responding to incoming ICMP echo requests received from the network**
- **Sending and ICMP echo requests and processing ICMP echo replies**

2.5.2.2 HsIcmp API

2.5.2.2.1 HsIcmpInit

Declaration:
extern int HsIcmpInit(hsicmp_init_t *init);
Summary:
This function is used to initialize ICMP library.

Parameters:
hsicmp_init_t *init – initialization structure defined as follows:

Variable	Description
hsicmp_timer_start_t *start_timer	start timer callback **Prototype:** **typedef long** hsicmp_timer_start_t **(unsigned long timeout_ms, void *arg, hsicmp_timer_cb_t *cb);** **timeout_ms – timeout in milliseconds** **arg – argument that HsIcmp passed to callback function** **cb – function to call on timer expiry** **Returns timer handle** **Timer expiry callback prototype:** **typedef void hsicmp_timer_cb_t (void *arg);** **arg – argument same as passed to timer_start function**
hsicmp_timer_stop_t *stop_timer	stop timer callback

	Prototype: **typedef void** hsicmp_timer_stop_t **(long timer_han);** **time_han – timer handle**
hsicmp_event_fn_t *event_fn	event callback Ptototype: typedef long hsicmp_event_fn_t(int ev, long arg1, long arg2); See ICMP events for description
hsicmp_log_fn_t *log_fn	event log function callback Function used buy HsIp to log debug events Prototype: typedef void hsicmp_log_fn_t(char *str); **Prototype:** **typedef void** hsicmp_log_fn_t **(char *str);** **str – event string to print or to put into event log**
hsicmp_get_ticks_fn_t *get_ticks	Callback function to get number of millisecond ticks since bootup Prototype: **typedef unsigned long hsicmp_get_ticks_fn_t(void);** **returns number of millisecond ticks since bootup**

Return values:

- **HS_ICMP_RC_OK – success**
- **HS_ICMP_RC_BAD_PARAM – Invalid parameter(s)**
- **HS_ICMP_RC_ALR_INIT – Library already initialized**

Sample usage:

```
icmp_init.event_fn = init->icmp_event_fn;
icmp_init.get_ticks = init->get_ticks;
icmp_init.log_fn = init->log_fn;
icmp_init.start_timer = init->timer_start;
icmp_init.stop_timer = init->timer_stop;
HsIcmpInit(&icmp_init);
```

2.5.2.2.2 HsIcmpCleanUp

Declaration:
extern void HsIcmpCleanUp(void);

Summary:
This function is used to de- initialize ICMP library.

Parameters:
none

len – length of buffer in bytes

Return values:
none

Sample usage:

HsIcmpCleanUp();

2.5.2.2.3 HsIcmpPing

Declaration:
extern int HsIcmpPing(unsigned long dest_ip, unsigned long timeout_ms, int num_pings, unsigned short data_size);

Summary:
This function is used to ping remote host using ICMP echo request packets.

Parameters:
dest_ip – destination IP address to ping

timeout_ms – timeout in millisecond for each ICMP ping (time to wait for each ICMP echo reply)

num_pings - number of times to ping

data_size – number of data bytes to put into ICMP Echo request into data portion of the packet

Return values:
- **HS_ICMP_RC_OK – success, ping initiated**
- **HS_ICMP_RC_BAD_PARAM – Invalid parameter(s)**
- **HS_ICMP_RC_NOT_INIT – Library not initialized**
- **HS_ICMP_RC_BUSY – Ping operation is already in progress**

Sample usage:

ping_rc = HsIcmpPing(addr, timeout_ms, num_pings, ping_size);

2.5.2.2.4 HsIcmpCancelPing

Declaration:
extern void HsIcmpCancelPing(void);

Summary:
This function is used to cancel current ICMP ping in progress.

Parameters:
none

Return values:
none

Sample usage:

HsIcmpCancelPing();

2.5.2.2.5 HsIcmpReceivePacket

Declaration:
extern void HsIcmpReceivePacket(unsigned long from_ip, unsigned char *pIcmpPkt, int length);

Summary:
This function is used to pass a received ICMP packet to HsIcmp library for further processing

Parameters:

from_ip – IP address from which ICMP packet was received from

length – length of ICMP packet in bytes

Return values:
none

Sample usage:

```
/* process reassembled datagram */
static void hsip_process_complete_ip_datagram(
        unsigned char   *pIpPkt,// ip packet
        unsigned short  total_len,      // ip packet data length (not including header)
        unsigned char   protocol,       // protocol
        unsigned long   src_ip          // source ip address
)
{
        switch (protocol)
        {
        case HS_IP_UDP_PROTO:
                HsUdpReceivePacket(pIpPkt, (int)total_len);
                break;

        case HS_IP_TCP_PROTO:
                HsTcpReceivePacket(pIpPkt, total_len);
                break;

        case HS_IP_ICMP_PROTO:
                HsIcmpReceivePacket(src_ip, pIpPkt, (int)total_len);
                break;
```

```
        default:
                return;
        }
}
```

2.5.2.3 HsIcmp Events

Event	Description
HS_ICMP_EV_PING_TIMEOUT	ICMP echo request timed out Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_SENT	ICMP echo request sent Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_REPLY	ICMP echo reply received Arg1= icmp sequence number Arg2 = 0
HS_ICMP_EV_PING_FINISHED	HsIcmp finished ping sequence: Arg1= number of ping requests sent Arg2 = number of ping replies received

2.5.3 HsArp

2.5.3.1 Overview

HsArp module implements ARP (Address Resolution Protocol) protocol as specified in RFC 826. HsArp primary functions include:

- **Serving HsIP and HsIcmp requests (via API call) to resolve 4 byte IP address to a 6 byte MAC address.**
- **Building and sending ARP protocol packets**
- **Reception of processing of ARP protocol packets**
- **HsArp maintains ARP table and uses fast hash lookup algorithm to find MAC address by IP address if it is already in ARP table**

2.5.3.2 HsArp API

2.5.3.2.1 HsArpInit

Declaration:
extern void HsArpInit(unsigned long local_ip, unsigned char *local_eth, hsarp_drv_tx_fn_t *drv_tx, hsarp_log_fn_t *log_fn);

Summary:
This function is used to initialize HsArp library

Parameters:
unsigned long local_ip – local IP address, 32 bit unsigned integer

unsigned char *local_eth – pointer to 6 byte buffer representing local hardware address (Ethernet MAC address)

hsarp_drv_tx_fn_t *drv_tx – pointer to callback function for HsArp to use to transmit ARP packets, defined as:

typedef void hsarp_drv_tx_fn_t(unsigned char *pkt, int len);

pkt – ARP packet pointer
len – length of ARP packet in bytes

hsarp_log_fn_t *log_fn – pointer to callback function for HsArp to use to log debug events, defined as:

typedef void hsarp_log_fn_t(char *str);

str –string to print or to put into event log.

Return values:
none

Sample usage:

HsArpInit(init->local_ip, init->local_eth_addr, init->drv_tx, init->log_fn);

2.5.3.2.2 HsArpSetParams

Declaration:
extern void HsArpSetParams(hs_arp_params_t *pPar);

Summary:
This function is used to set updated IP parameters to HsArp library. It is used to inform HsArp library of new IP address received from DHCP server

Parameters:

hs_arp_params_t *pPar – pointer to parameter structure as follows:

```
typedef struct
{
        unsigned long local_ip;                 // local ip address
} hs_arp_params_t;
```

Return values:
none

Sample usage:

HsArpSetParams(&sArpPar);

2.5.3.2.3 HsArpResolveAddress

Declaration:
extern int HsArpResolveAddress(unsigned long dest_ip, unsigned char *eth_addr);

Summary:
This function is used to find Ethernet MAC address corresponding to give IP address. This function will first lookup local ARP table and if a match is found the function returns the result MAC address immediately. Otherwise it initiates ARP request and returns FALSE; user must call this function again at a later time (allowing ARP reply packet to arrive and be processed by HsArp).

Parameters:

dest_ip – IP address of the host to lookup

*eth_addr – pointer to 6 byte buffer to receive Ethernet MAC address of the host with given IP address

Return values:
1 – success, MAC address copied into *eth_addr
0 – fail, call back later

Sample usage:

mac_determined = HsArpResolveAddress(dest_ip, destmac);

2.5.3.2.4 HsArpReceivedEthPacket

Declaration:

extern void HsArpReceivedEthPacket(unsigned char *pkt, int len);

Summary:

This function is used to pass a received ARP packet to HsArp library for further processing

Parameters:

pkt – pointer to start of ARP packet

len – length of ARP packet in bytes

Return values:

none

Sample usage:

```
/* Ethernet packet received, not thread safe, MUST NOT be called directly from Interrupt */
void HsIpReceivePacket(unsigned char *pkt, int len)
{
	unsigned short ethernet_type;

	if (!ip.initialized)
		return;

	if ((!pkt) || (!len))
		return;

	// discard frames with size less than ethernet header
	if (len < HS_ETH_HDR_SZ)
		return;

	get_word(&pkt[12], &ethernet_type);

	// check for types of frames we process,
	switch (ethernet_type)
	{
	case HS_ETHTYPE_IP:
		hsip_receive_eth_packet(pkt, len);
		break;

	case HS_ETHTYPE_ARP:
		HsArpReceivedEthPacket(pkt, len);
		break;

	// discard protocols we don't support
	default:
		return;
	}
}
```