

# HS FTP Client C Source Library v1.3.5 User Manual

Revision: 1.8  
Date: 13 February 2012

<b><u>HS FTP CLIENT C SOURCE LIBRARY</u></b>	
<b><u>V1.3.5 USER MANUAL.....</u></b>	<b><u>1</u></b>
<b><u>REVISION: 1.8.....</u></b>	<b><u>1</u></b>
<b><u>DATE: 13 FEBRUARY 2012.....</u></b>	<b><u>1</u></b>
<b><u>1 INTRODUCTION.....</u></b>	<b><u>3</u></b>
<b><u>1.1 OVERVIEW.....</u></b>	<b><u>3</u></b>
<b><u>1.2 APPLICATION MODEL.....</u></b>	<b><u>4</u></b>
<b><u>2 HS FTP API.....</u></b>	<b><u>5</u></b>
<b><u>2.1 HsFTPINIT.....</u></b>	<b><u>5</u></b>
<b><u>2.2 HsFTPCLEANUP.....</u></b>	<b><u>8</u></b>
<b><u>2.3 HsFTPCLIOPEN.....</u></b>	<b><u>8</u></b>
<b><u>2.4 HsFTPCLICLOSE.....</u></b>	<b><u>8</u></b>
<b><u>2.5 HsFTPCLICONNECT.....</u></b>	<b><u>9</u></b>
<b><u>2.6 HsFTPCLIDISCONNECT.....</u></b>	<b><u>10</u></b>
<b><u>2.7 HsFTPCLIHDR.....</u></b>	<b><u>10</u></b>
<b><u>2.8 HsFTPCLICREATEDIR.....</u></b>	<b><u>11</u></b>
<b><u>2.9 HsFTPCLIREMOVEDIR.....</u></b>	<b><u>11</u></b>
<b><u>2.10 HsFTPCLILIST.....</u></b>	<b><u>12</u></b>
<b><u>2.11 HsFTPCLIGETFILE.....</u></b>	<b><u>12</u></b>
<b><u>2.12 HsFTPCLISENDFILE.....</u></b>	<b><u>13</u></b>
<b><u>2.13 HsFTPCLIDELETEFILE.....</u></b>	<b><u>13</u></b>
<b><u>2.14 HsFTPCLIABORT.....</u></b>	<b><u>14</u></b>
<b><u>2.15 HsFTPCLIRENAME.....</u></b>	<b><u>14</u></b>
<b><u>2.16 HsFTPCLIGETCURRENTDIRECTORY.....</u></b>	<b><u>15</u></b>
<b><u>2.17 HsFTPSETCONFIG.....</u></b>	<b><u>15</u></b>
<b><u>2.18 HsFTPCLIGETEVENT.....</u></b>	<b><u>16</u></b>
<b><u>3 HS FTP CLIENT EVENTS.....</u></b>	<b><u>17</u></b>
<b><u>4 RECURSIVE FOLDER OPERATIONS.....</u></b>	<b><u>20</u></b>
<b><u>4.1 API FUNCTIONS.....</u></b>	<b><u>20</u></b>
<b><u>4.2 HsFTPRECURSINIT.....</u></b>	<b><u>20</u></b>
<b><u>4.3 HsFTPRECURSECLEANUP.....</u></b>	<b><u>20</u></b>

<a href="#">4.4 HsFTPRecursTick.....</a>	<a href="#">21</a>
<a href="#">4.5 HsFTPRecursDownloadFolder.....</a>	<a href="#">21</a>
<a href="#">4.6 HsFTPRecursUploadFolder.....</a>	<a href="#">22</a>
<a href="#">4.7 HsFTPRecursDeleteFolder.....</a>	<a href="#">22</a>
<a href="#">4.8 RECURSIVE OPERATIONS CALLBACK AND EVENTS.....</a>	<a href="#">23</a>
<a href="#">4.9 RECURSIVE OPERATIONS MODULE RETURN CODES.....</a>	<a href="#">24</a>

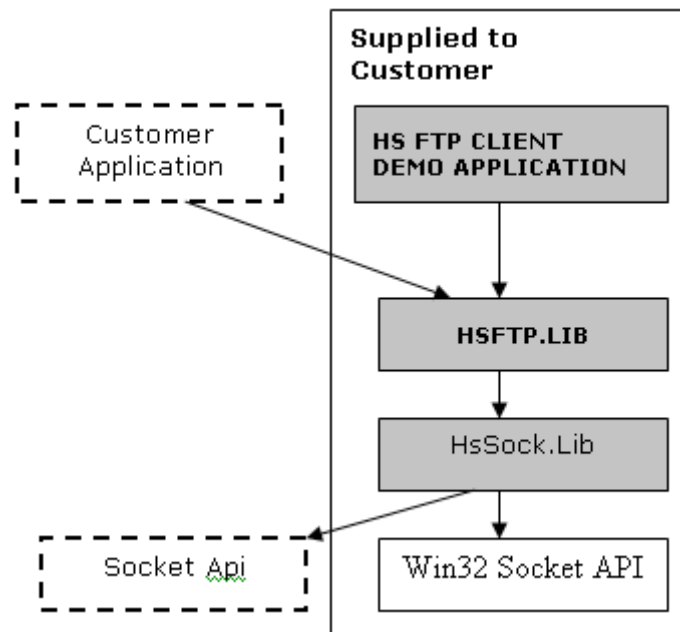
# 1 Introduction

## 1.1 Overview

HS FTP is a software library in C (supplied with full source code) which 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.

The library architecture is shown in the diagram below:



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
- "PWD" – request current directory name
- "RNFR" – rename from
- "RNTD" – 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)

## **1.2 Application Model**

To use HS FTP Library an application should generally follow these steps:

- Initialize HS FTP Library only once at application startup from the main thread or from one of threads in a multi-threaded application by calling `HsFtpInit()`
- Open an FTP session context and obtain an FTP session handle (call `HsFtpCliOpen()` )
- Once the FTP session handle is open, the application should continually poll HS FTP for events associated with that handle by calling `HsFtpCliGetEvent()`. It is important to realize that periodic calling of `HsFtpCliGetEvent()` not only empties the event queue that the HS FTP fills, but also drives internal operations, such as receiving data from underlying socket layer and driving internal timers.
- Connect to an FTP server by calling `HsFtpCliConnect()`, once the code that is continually polling the HS FTP Library for events detects event `HS_FTPCLI_USR_EV_LOGGEDIN`, the connection to the sever has been established.
- Call any session related functions to do work while connected to the server, for example `HsFtpCliSendFile()`, `HsFtpCliGetFile()`, `HsFtpCliList()` and so on.
- Disconnect from FTP server by calling `HsFtpCliDisconnect()`, the code polling for events by calling `HsFtpCliGetEvent()`, detects event `HS_FTPCLI_USR_EV_CLOSED`, move to next step.
- Close and deallocate the FTP session context by calling `HsFtpCliClose()`
- Before exiting the application de-initialize the HS FTP Library by calling `HsFtpCleanUp()` from the main thread or from one of other threads in a multi-threaded application.

The HS FTP Library is designed to be thread safe, several open sessions may be opened, closed and operated concurrently from multiple threads. The current version supports a maximum of 32 concurrent FTP sessions.

## 2 HS FTP API

### 2.1 HsFtpInit

int HsFtpInit(hs_ftp_init_t *pInit)		
<b>DESCRIPTION</b>		
<i>This function initialises HS FTP Library and MUST be called once, prior to calling any other library functions</i>		
<b>PARAMETERS</b>		
Type	Name	Description
hs_ftp_init_t	*pInit	<p>Pointer to initialization structure defined as follows:</p> <pre>// init structure typedef struct {     hsftp_create_file_t *create_file;    // Create file callback     hsftp_close_file_t *close_file;     // Close file callback     hsftp_write_file_t *write_file;     // Write file callback     hsftp_read_file_t *read_file;      // Read file callback     int log_enabled;                   // enables debug event logging     log_fn_t *log_fn;                  // log event callback     hs_ftp_get_ticks_fn_t *get_ms_ticks; // millisecond tick callback } hs_ftp_init_t;</pre> <p>The function prototypes, parameters and return values are defined in the following section</p>
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_ALRINIT – library is already initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters: either pInit is NULL or one or more of function pointers of hs_ftp_init_t structure are NULL</i></li> <li>• <i>HS_FTPCLI_RC_SOCKETINIT_FAIL – socket layer initialisation failed</i></li> <li>• <i>HS_FTP_RC_OK – success</i></li> </ul>		

#### 2.1.1 Structure hs\_ftp\_init\_t definition

```
// init structure
typedef struct
{
    hsftp_create_file_t *create_file;    // Create file callback
    hsftp_close_file_t *close_file;     // Close file callback
    hsftp_write_file_t *write_file;     // Write file callback
    hsftp_read_file_t *read_file;      // Read file callback
    int log_enabled;                   // enables debug event logging
    log_fn_t *log_fn;                  // log event callback
} hs_ftp_init_t;
```

Callback function	Description
<b>create_file</b>	<p>Pointer to a function in user code that will be called by HSFTP when it need to create or open an existing disk file. Function prototype is defined as:</p> <pre>void *hsftp_create_file_t(unsigned char *filename, int mode, int open_always);</pre> <p>filename – pointer to null terminated filename string of the file to open or create  mode – one of the following:  HSFTP_FILE_READ – open for reading  HSFTP_FILE_WRITE – open for writing  open_always – 1 = if a file does not exist a new file is created; 0 = a file must exist or the function will fail</p> <p>Returns: a file handle if open/ create successful or NULL otherwise</p>
<b>close_file</b>	<p>pointer to a function in user code that will be called by HSFTP when it need to close a previously opened disk file. Function prototype is defined as:</p> <pre>typedef void hsftp_close_file_t(void *hFile);</pre> <p>hFile – file handle returned by callback create_file</p>

<b>write_file</b>	<p>pointer to a function in user code that will be called by HSFTP when it need to write a block of data into a disk file. Function prototype is defined as:</p> <pre>typedef int hsftp_write_file_t(void *hFile, void *pData, int length, int *bytes_written);</pre> <p>hFile – file handle returned by callback create_file          pData – pointer to data buffer to write          length – length of data buffer in bytes to write          *bytes_written – number of bytes actually written to be returned in this variable</p> <p>Returns 1 = success; 0 = write operation failed</p>
<b>read_file</b>	<p>pointer to a function in user code that will be called by HSFTP when it need to read a block of data from a disk file. Function prototype is defined as:</p> <pre>typedef int hsftp_read_file_t(void *hFile, void *pData, int length, int *bytes_read);</pre> <p>hFile – file handle returned by callback create_file          pData – pointer to data buffer to read data into          length – length of data buffer in bytes to read          *bytes_read – number of bytes actually read to be returned in this variable</p> <p>Returns 1 = success; 0 = read operation failed</p>
<b>log_enabled</b>	Enables HS FTP to log debug events, such as extra error information, state transitions, event processing in FTP FSM.
<b>log_fn</b>	<p>Event logging callback. If log_enabled = 1, HS FTP calls this function to log events. Prototype is as follows:</p> <pre>typedef void log_fn_t(char *str);</pre> <p>str – null terminated event string to be printed in the application event log</p>
<b>get_ms_ticks</b>	<p>Used by HS FTP to get current system millisecond tick</p> <p>Prototype:</p> <pre>typedef unsigned long hs_ftp_get_ticks_fn_t(void);</pre>

## 2.1.2 Sample implementation of function callbacks defined in hs\_ftp\_init\_t

This example assumes Windows OS, but similar functions may be implemented for other operating systems:

```

/*
 * 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
 */
static void hstfp_close_file(void *hFile)
{
    CloseHandle((HANDLE)hFile);
}

/*
 * Read file callback
 */
static int hstfp_read_file(void *user_ref, 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
 */
static 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;
}

/*
 * write event into event log
 */
static void write_event(char *str)
{
    PutLog(dlg_main, str);
}

// get millisecond tick
static unsigned long hsftp_get_ticks(void)
{
    unsigned long res;

    res = timeGetTime();

    return res;
}

// Initialize HsFtp
hs_ftp_init_t init = {0};
init.close_file = hstfp_close_file;
init.create_file = hstfp_create_file;
init.read_file = hstfp_read_file;
init.write_file = hstfp_write_file;
init.log_fn = write_event;
init.log_enabled = 1;
init.get_ms_ticks = hsftp_get_ticks;

HsFtpInit(&init);
```

## 2.2 HsFtpCleanUp

int HsFtpCleanUp (void)		
<b>DESCRIPTION</b>		
<i>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</i>		
<b>PARAMETERS</b>		
Type	Name	Description
None	None	None
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTP_RC_OK – success</i></li> </ul>		

## 2.3 HsFtpCliOpen

int HsFtpCliOpen (void)		
<b>DESCRIPTION</b>		
<i>This function opens an FTP context and gets a new session handle. This function must be called before calling any session related functions (e.g before HsFtpCliConnect, HsFtpGetFile, etc). The session must be closed / de-allocated by calling HsFtpCliClose.</i>		
<b>PARAMETERS</b>		
Type	Name	Description
None	None	None
<b>RETURNS</b>		
<i>Returns a non-negative integer session handle or a negative error code, one of the following:</i>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_NO_CTX – cannot allocate a new session, reached maximum number of concurrent sessions</i></li> </ul>		

## 2.4 HsFtpCliClose

int HsFtpCliClose (int session_handle)		
<b>DESCRIPTION</b>		
<i>This function de-allocates an FTP context. Please note that this function only de-allocates the session. An application must call HsFtpCliDisconnect to disconnect / log out from the FTP server and then call HsFtpCliClose on processing the HS_FTPCLI_USR_EV_CLOSED event.</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	Session handle obtained earlier by calling HsFtpCliOpen
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTP_RC_OK – success, session closed</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – bad handle value</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – session is not allocated</i></li> <li>• <i>HS_FTPCLI_RC_INVALID_CMD – the session is not in a valid state to be closed. If the session is connected to the FTP server (or in the process of connecting), the application must call HsFtpCliDisconnect and wait for HS_FTPCLI_USR_EV_CLOSED.</i></li> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> </ul>		

## 2.5 HsFtpCliConnect

```
extern int HsFtpCliConnect(int session_handle, hs_ftp_conn_t *conn)
```

### DESCRIPTION

*This function is used to connect to remote FTP server*

### PARAMETERS

Type	Name	Description
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.
int	session_handle	Session handle obtained earlier by calling HsFtpCliOpen

### RETURNS

- *HS\_FTPCLI\_RC\_NOTINIT – library not initialised*
- *HS\_FTPCLI\_RC\_INV\_PAR – invalid parameters supplied*
- *HS\_FTPCLI\_RC\_NOTALLOC – session not allocated*
- *HS\_FTPCLI\_RC\_INVALID\_CMD – session is not in a valid state for this operation*
- *HS\_FTPCLI\_RC\_TCPCONNFAIL – outgoing TCP connection to remote FTP server failed*
- *HS\_FTP\_RC\_OK – success, HS FTP module started session establishment to remote FTP server. The application must continuously poll for events using function HsFtpCliGetEvent*

### 2.5.1 Description of hs\_ftp\_conn\_t structure

Data Member	Description
unsigned char *srv_name;	Remote FTP server DNS name to connect to, for example ( <a href="http://ftp.hillstone-software.com">ftp.hillstone-software.com</a> ). 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 *username;	FTP account user name for authentication, pointer to null terminated string
unsigned char *password;	FTP account password for authentication, pointer to null terminated string

## 2.6 HsFtpCliDisconnect

extern int HsFtpCliDisconnect(long session_handle)		
<b>DESCRIPTION</b>		
<i>This function is used to close FTP session to remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTP_RC_OK – success, session disconnect initiated. When the session is closed, the application receives event HS_FTPCLI_USR_EV_CLOSED and at that point it is safe to call HsFtpCliClose to deallocate the session context.</i></li> </ul>		

## 2.7 HsFtpCliChDir

extern int HsFtpCliChDir(int session_handle, unsigned char *dir)		
<b>DESCRIPTION</b>		
<i>This function is used to change to a different directory on remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
unsigned char	*dir	Directory name to change to, null terminated string
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTPCLI_RC_INVALID_CMD – command is invalid for current session state</i></li> <li>• <i>HS_FTP_RC_OK – success, HS FTP initiated procedure to change to a new directory on remote FTP server. The actual result shall be asynchronously indicated via user event HS_FTPCLI_USR_EV_CWDDONE</i></li> </ul>		

## 2.8 HsFtpCliCreateDir

extern int HsFtpCliCreateDir(int session_handle, unsigned char *dir)		
<b>DESCRIPTION</b>		
<i>This function is used to create a new directory on remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliConnect call.
unsigned char	*dir	Directory name, null terminated string
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTPCLI_RC_INVALID_CMD – command is invalid for current session state</i></li> <li>• <i>HS_FTP_RC_OK – success, HS FTP initiated procedure to create a new directory on remote FTP server. The actual result shall be asynchronously indicated via event HS_FTPCLI_USR_EV_MKDDONE</i></li> </ul>		

## 2.9 HsFtpCliRemoveDir

extern int HsFtpCliRemoveDir(int session_handle, unsigned char *dir)		
<b>DESCRIPTION</b>		
<i>This function is used to remove a directory on remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
unsigned char	*dir	Directory name, null terminated string
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTPCLI_RC_INVALID_CMD – command is invalid for current session state</i></li> <li>• <i>HS_FTP_RC_OK – success, HS FTP initiated procedure to remove a directory on remote FTP server. The actual result shall be asynchronously indicated via event HS_FTPCLI_USR_EV_RMDDONE</i></li> </ul>		

## 2.10 HsFtpCliList

```
extern int HsFtpCliList(int session_handle);
```

### DESCRIPTION

This function is used to receive directory file listing from the remote FTP server. The listing is returned asynchronously via event `HS_FTPCLI_USR_EV_LISTDONE`, please see the detailed description in section 3 (HS FTP Client Module to USER 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

Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.

### RETURNS

- `HS_FTPCLI_RC_NOTINIT` – library not initialised
- `HS_FTPCLI_RC_INV_PAR` – invalid parameters supplied
- `HS_FTPCLI_RC_NOTALLOC` – invalid session – not allocated
- `HS_FTP_RC_OK` – success, HS FTP library initiated a procedure to receive directory listing from remote FTP server. The actual result shall be asynchronously indicated via event `HS_FTPCLI_USR_EV_LISTDONE`

## 2.11 HsFtpCliGetFile

```
extern int HsFtpCliGetFile(int session_handle, unsigned char *filename)
```

### DESCRIPTION

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 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

Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
unsigned char	*filename	Filename to get, pointer to null terminated string

### RETURNS

- `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
- `HS_FTP_RC_OK` – success, HS FTP initiated procedure to start downloading the specified file from remote FTP server. The actual result shall be indicated via `HS_FTPCLI_USR_EV_RXDONE` event

## 2.12 HsFtpCliSendFile

extern int HsFtpCliSendFile (int session\_handle, unsigned char \*filename)

### DESCRIPTION

*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

Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
unsigned char	*filename	Filename to get, pointer to null terminated string

### RETURNS

- *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*
- *HS\_FTP\_RC\_OK – success, HS FTP initiated procedure to start uploading the specified file to remote FTP server. The completion of file transfer shall be indicated via user via HS\_FTPCLI\_USR\_EV\_TXDONE event*

## 2.13 HsFtpCliDeleteFile

extern int HsFtpCliDeleteFile(int session\_handle, unsigned char \*filename)

### DESCRIPTION

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

### PARAMETERS

Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
unsigned char	*filename	File name, null terminated string

### RETURNS

- *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*
- *HS\_FTP\_RC\_OK – success, HS FTP initiated procedure to delete a file on remote FTP server. The actual result shall be asynchronously indicated via user event HS\_FTPCLI\_USR\_EV\_DELDONE*

## 2.14 HsFtpCliAbort

extern int HsFtpCliAbort (int session_handle)		
<b>DESCRIPTION</b>		
<i>This function is used to abort current FTP operation in progress</i>		
<b>PARAMETERS</b>		
Type	Name	Description
long	session_handle	HS FTP session handle, returned in HsFtpCliOpen call.
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTP_RC_OK – success, HS FTP initiated the procedure to abort current command. The completion shall be asynchronously indicated via event HS_FTPCLI_USR_EV_ABORTED</i></li> </ul>		

## 2.15 HsFtpCliRename

extern int HsFtpCliRename(int session_handle, unsigned char *oldname, unsigned char *newname)		
<b>DESCRIPTION</b>		
<i>This function is used to rename a file or folder on remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliOpen 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
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HS_FTPCLI_RC_NOTINIT – library not initialised</i></li> <li>• <i>HS_FTPCLI_RC_INV_PAR – invalid parameters supplied</i></li> <li>• <i>HS_FTPCLI_RC_NOTALLOC – invalid session – not allocated</i></li> <li>• <i>HS_FTPCLI_RC_INVALID_CMD – command is invalid for current session state</i></li> <li>• <i>HS_FTP_RC_OK – success, HS FTP initiated procedure to rename a file or folder on remote FTP server. The actual result shall be asynchronously indicated via user event HS_FTPCLI_USR_EV_RENDONE</i></li> </ul>		

## 2.16 HsFtpCliGetCurrentDirectory

```
extern int HsFtpCliGetCurrentDirectory(int session_handle);
```

### DESCRIPTION

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

### PARAMETERS

Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliConnect call.

### RETURNS

- *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*
- *HS\_FTP\_RC\_OK – success, HS FTP initiated procedure to request the current working directory from remote FTP server. The actual result shall be asynchronously indicated via user event HS\_FTPCLI\_USR\_EV\_PWDDONE*

## 2.17 HsFtpSetConfig

```
extern void HsFtpSetConfig (hs_ftp_config_t *cfg);
```

### DESCRIPTION

*This function is used to set HS FTP global configuration parameters*

### PARAMETERS

Type	Name	Description
hs_ftp_config_t	*cfg	Pointer to configuration structure hs_ftp_config_t, with the following data members:  long t1_timeout – timeout in milliseconds for all HS FTP operations. Default timeout is 15000 milliseconds  int log_enabled – enable or disable debug event logging via log_fn callback (see HsFtpInIt)

### RETURNS

*NONE*

## 2.18 HsFtpCliGetEvent

```
int HsFtpCliGetEvent(int session_handle, int *ev, long *arg1, long *arg2, long *arg3, char *strBuf, int *buflen);
```

### DESCRIPTION

*This function is used to poll HS FTP for events. The user application must call this function as often as possible for each open FTP session.*

### PARAMETERS

Type	Name	Description
int	session_handle	FTP session handle obtained by calling HsFtpCliOpen
int	*ev	Pointer to integer to receive event id
long	*arg1	Pointer to long to receive first argument
long	*arg2	Pointer to long to receive second argument
long	*arg3	Pointer to long to receive third argument
char	*strBuf	Pointer to character buffer of 2048 bytes to receive a message string
int	*buflen	Pointer to integer variable indicating the length of data contained in the strBuf

### RETURNS

- *HS\_FTPCLI\_RC\_NOTINIT – library not initialized*
- *HS\_FTPCLI\_RC\_INV\_PAR – invalid parameters*
- *HS\_FTPCLI\_RC\_NOTALLOC – session not allocated*
- *HS\_FTPCLI\_RC\_NO\_EVENTS – no events available*
- *HS\_FTP\_RC\_OK – an event is available. Event id is copied into \*ev and arguments are copied to \*arg1, \*arg2, \*arg3, an error / information text may be copied to strBuf and the length buflen may be set. If a text is available, the buflen is non zero.*

### 3 HS FTP Client Events

Event	Description
HS_FTPCLI_USR_EV_LOGGEDIN	<p>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.</p> <p>FTP server reply (for example "226 logged in") may be present in strBuf and the buflen is set to the length of the reply string</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p>
HS_FTPCLI_USR_EV_CLOSED	<p>FTP control connection and FTP session closed</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply or other error string may be present in strBuf and the buflen is set to the length of the buffer string.</p> <p>On receiving this event the applicatoin may de-allocate the FTP session by calling HsFtpCliClose or call HsFtpCliConnect to establish new connection using the same FTP session context</p>
HS_FTPCLI_USR_EV_LISTDONE	<p>This event is used by HS FTP library to return a buffer containing current directory listing from remote FTP server after HsFtpCliList function call</p> <p>Arg1 – pointer to start of buffer containing remote FTP server current directory listing. Arg2 – length of buffer Arg3 – 0</p> <p>The memory containing the listing data pointed to by Arg1 is allocated by HS FTP from program heap using malloc(). The application must free the memory after it no longer needs it by calling free(arg1);</p>
HS_FTPCLI_USR_EV_CWDDONE	<p>HsFtpCliChDir function successfully completed – Current directory on remote FTP server successfully changed to new specified directory.</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_RXDONE	<p>File successfully received – HsFtpCliGetFile function completed.</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_TXDONE	<p>File successfully transmitted – HsFtpCliSendFile completed.</p> <p>Arg1 = 0 Arg2 = 0</p>

	<p>Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_TX_STATUS	<p>Notifies the application that the next block of file data has been transmitted</p> <p>Arg1 – size of data block in bytes that has been transmitted</p>
HS_FTPCLI_USR_EV_RX_STATUS	<p>Notifies the application that the next data block of file from remote FTP server has been received</p> <p>Arg1 – size of data block in bytes that has been received</p>
HS_FTPCLI_USR_EV_ABORTED	<p>Current FTP operation aborted – HsFtpCliAbort function completed</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_MKDDONE	<p>HsFtpCliCreateDir function successfully completed – Directory on remote FTP server successfully created.</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_RMDDONE	<p>HsFtpCliRemoveDir function successfully completed – Directory on remote FTP server successfully deleted</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_DELDONE	<p>HsFtpCliDeleteFile function successfully completed – File on remote FTP server successfully deleted</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>
HS_FTPCLI_USR_EV_RENDONE	<p>HsFtpCliRename function successfully completed file or folder on remote FTP server renamed</p> <p>Arg1 = 0 Arg2 = 0 Arg3 = 0</p> <p>FTP server reply string may be present in strBuf and the buflen is set to the length of the buffer string.</p>

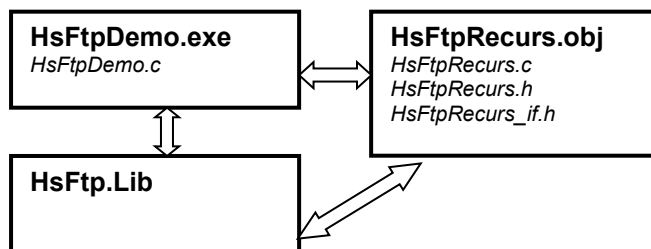
HS_FTPCLI_USR_EV_PWDDONE	HsFtpCliGetCurrentDirectory function completed successfully.  Arg1 = 0 Arg2 = 0 Arg3 = 0  strBuf contains the current path string buflen is set to the length of the current path string
--------------------------	---

## 4 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



### 4.1 API Functions

#### 4.2 HsFtpRecursInit

Extern void HsFtpRecursInit (hsftp\_recursive\_callback\_t \*cb)

#### DESCRIPTION

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

#### PARAMETERS

Type	Name	Description
hsftp_recursive_callback_t	*cb	Function pointer to recursive module callback functions which receives event notifications of folder operations completion, failures, status and progress. See section 4.7 for definition of the callback function

#### RETURNS

**NONE**

#### 4.3 HsFtpRecurseCleanUp

Extern void HsFtpRecurseCleanUp (void)

#### DESCRIPTION

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

#### PARAMETERS

**NONE**

#### RETURNS

**NONE**

#### 4.4 HsFtpRecursTick

void HsFtpTick(void)		
<b>DESCRIPTION</b>		
<i>This function must be called as often as possible from the main program loop.</i>		
<b>PARAMETERS</b>		
Type	Name	Description
None	None	None
<b>RETURNS</b>		
NONE		

#### 4.5 HsFtpRecursDownloadFolder

extern int HsFtpRecursDownloadFolder (int session_handle, unsigned char *foldername, int overwrite)		
<b>DESCRIPTION</b>		
<i>This function is used to download an entire folder with all sub-folders and files from remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	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
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HSFTP_RECURS_RC_NOTINIT – Recursive folder operations module not initialised</i></li> <li>• <i>HSFTP_RECURS_RC_INVPAR – invalid parameters supplied</i></li> <li>• <i>HSFTP_RECURS_RC_BUSY – command is invalid for current session state</i></li> <li>• <i>HSFTP_RECURS_RC_OK – success, Recursive folder download started. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in section 4.8</i></li> </ul>		

#### 4.6 HsFtpRecursUploadFolder

extern int HsFtpRecursUploadFolder (int session_handle, unsigned char *foldername)		
<b>DESCRIPTION</b>		
<i>This function is used to upload an entire folder with all sub-folders and files to remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliConnect call.
unsigned char	*foldername	Folder name to upload, null terminated string
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HSFTP_RECURS_RC_NOTINIT – Recursive folder operations module not initialised</i></li> <li>• <i>HSFTP_RECURS_RC_INVPAR – invalid parameters supplied</i></li> <li>• <i>HSFTP_RECURS_RC_BUSY – command is invalid for current session state</i></li> <li>• <i>HSFTP_RECURS_RC_LCHDIR – failed to change into local directory (local chdir failed)</i></li> <li>• <i>HSFTP_RECURS_RC_OK – success, Recursive folder upload started. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in section 4.8</i></li> </ul>		

#### 4.7 HsFtpRecursDeleteFolder

extern int HsFtpRecursUploadFolder (int session_handle, unsigned char *foldername)		
<b>DESCRIPTION</b>		
<i>This function is used to delete an entire folder with all sub-folders and files from remote FTP server</i>		
<b>PARAMETERS</b>		
Type	Name	Description
int	session_handle	HS FTP session handle, returned in HsFtpCliConnect call.
unsigned char	*foldername	Folder name to delete, null terminated string
<b>RETURNS</b>		
<ul style="list-style-type: none"> <li>• <i>HSFTP_RECURS_RC_NOTINIT – Recursive folder operations module not initialised</i></li> <li>• <i>HSFTP_RECURS_RC_INVPAR – invalid parameters supplied</i></li> <li>• <i>HSFTP_RECURS_RC_BUSY – command is invalid for current session state</i></li> <li>• <i>HSFTP_RECURS_RC_OK – success, Recursive folder deletion started. The actual result shall be asynchronously indicated via user callback event, please see the detailed description in section 4.8</i></li> </ul>		

## 4.8 Recursive Operations Callback and Events

### 4.8.1 Event Callback Prototype

```
typedef void hsftp_recursive_callback_t (int 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

### 4.8.2 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:

	<p>1 = Start of individual item upload 2 = Completion of individual item upload</p>
HSFTP_RECURS_USR_EV_FOLDER_DELETE_FAILED	<p>Recursive folder delete operation failed.</p> <p>Arg1 – Recursive operations module return code, see next section for a list of return codes</p> <p>Arg2 – Core HS FTP library event that caused this event, see section 3.2 for a list of core HS FTP events</p>
HSFTP_RECURS_USR_EV_FOLDER_DELETE_DONE	<p>Recursive folder delete operation complete with success.</p> <p>Arg1 – 0 Arg2 – 0</p>
HSFTP_RECURS_USR_EV_FOLDER_DELETE_STATUS	<p>reports start of stop of individual item delete operation</p> <p>Arg1 – If Arg2 == 1, long pointer to a pathname of the item now being deleted</p> <p>Arg2 – status code: 1 = Start of individual item delete 2 = Completion of individual item delete</p>
HSFTP_RECURS_USR_EV_FOLDER_PROGRESS	<p>Indicates individual item operation progress (item upload, download or delete)</p> <p>Arg1 – percent complete 0 – 100 Arg2 – 0</p>

#### 4.9 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_MKDIR	Failed to create folder at remote FTP server
HSFTP_RECURS_RC_FSEND	Failure during sending a file
HSFTP_RECURS_RC_PATH	Pathname too long