



HS TFTP .NET Class Library User Manual (v1.1)

Date: 10 August 2008

HS TFTP .NET Class Library User Manual (v1.1).....	1
1 Introduction	2
1.1 About HS TFTP .NET	2
1.2 HS TFTP .NET Architecture	2
2 API specification	3
2.1 HsTftpDllServerStart	3
2.2 HsTftpDllDestroy	4
2.3 HsTftpDllSetDirectory	4
2.4 HsTftpDllReadEvents.....	5
2.5 HsTftpDllGetSessionInfo	7
2.6 HsTftpDllGetSessionStats	9
2.7 HsTftpDllClientSendFile.....	10
2.8 HsTftpDllClientReceiveFile.....	11
2.9 HsTftpDllGetIPAddrFirst	12
2.10 HsTftpDllGetIPAddrNext	13
2.11 HsTftpDllSetIpIndex.....	14
2.12 Return Codes, Events and Other Definitions.....	15
2.12.1 HS TFTP .NET return codes	15
2.12.2 HS TFTP .NET event notifications	15
2.12.3 Other Definitions.....	15

1 Introduction

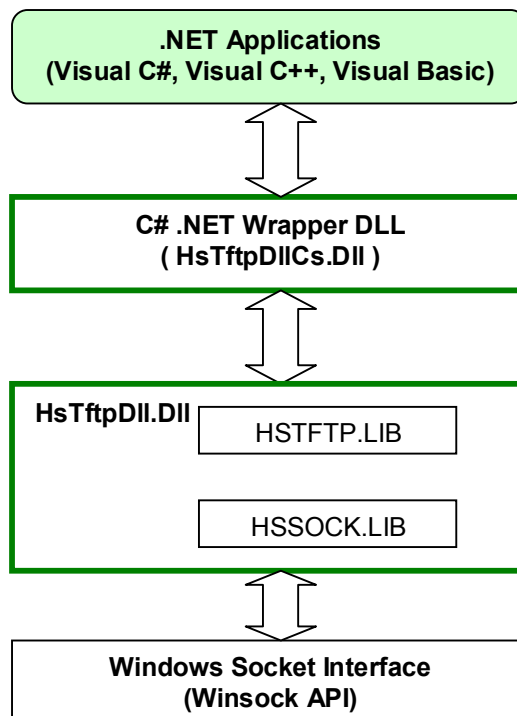
1.1 About HS TFTP .NET

HS TFTP .NET is a .NET class library, which implements Trivial File Transfer Protocol according to RFC1350. HS TFTP .NET supports simultaneous TFTP server and client operation.

HS TFTP .NET class library can be used from Visual C# .NET, Visual C++ and Visual Basic applications.

1.2 HS TFTP .NET Architecture

HS TFTP .NET class library internally consists of upper layer interface class DLL – HsTftpDIIcs.dll, core TFTP DLL – HsTftpDII.Dll, TFTP protocol module – HsTftp and Socket interface module HsSock.



2 API specification

2.1 HsTftpDllServerStart

Declaration:

```
[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllServerStart(
    int tftp_port,           // port number to start listening on
    String working_dir      // working directory
);
```

Summary:

Starts TFTP server, listening on specified TFTP port and in specified working directory

Parameters:

tftp_port

– IP port to listen on, default port for TFTP server is 69.

working_dir

– Working directory where files are to be received and sent from.

Return values:

HS_TFTP_RC_OK	Success, server started
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FAIL	Failed to start the server because Server already running
HS_TFTP_RC_SETDIR	Server not started. Failed to set working directory
HS_TFTP SOCK_STARTUP_FAIL	Socket interface failed to startup
HS_TFTP_RC_NO_MEM	Not enough memory
HS_TFTP_RC_UDP_SOCKET_OPEN	Failed to open UDP socket on specified port

Sample usage:

```
using HsTftpDllCs;
```

```
int rc;
```

```
rc = HsTftp.HsTftpDllServerStart(69, "c:\\");
if (rc == HS_TFTP_RC_OK)
{
    // success
}
else
{
    // failure
}
```

2.2 HsTftpDllDestroy

Declaration:

```
[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllDestroy();
```

Summary:

Stops the server if running, closes down all sessions and releases all resources used by HS TFTP .NET class library.

Parameters:

none

Return values:

HS_TFTP_RC_OK	Success
---------------	---------

Sample usage:

```
using HsTftpDllCs;

HsTftp.HsTftpDllDestroy();
```

2.3 HsTftpDllSetDirectory

Declaration:

```
[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllSetDirectory(
    String working_dir // working directory
);
```

Summary:

Sets current working directory for server mode file transfers

Parameters:

working_dir – working directory

Return values:

HS_TFTP_RC_OK	Success
HS_TFTP_RC_FAIL	Could not set working directory

Sample usage:

```
using HsTftpDllCs;

int rc;

rc = HsTftp.HsTftpDllSetDirectory("c:\\");
```

2.4 HsTftpDllReadEvents

Declaration:

```
[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllReadEvents(
    ref int evt,      // event
    ref int arg      // argument
);
```

Summary:

This method is intended to be called from the application based on a timer, recommended 10 ms to read events from HS TFTP .NET event queue. The events tell the calling application about new TFTP sessions that has been established, or existing TFTP transfers that have completed with success or failure.

Parameters:

evt – event variable, where next read event is to be stored, passed by reference
arg – argument variable, passed by reference

Possible event and argument values are listed below:

Event	Description
TFTPDLL_NO_EVENTS	Event queue empty, there are no new events
TFTPDLL_EV_FILE_RX_START	File receive operation started, arg contains new session handle
TFTPDLL_EV_FILE_TX_START	File send operation started, arg contains new session handle
TFTPDLL_EV_FILE_RX_COMPLETE	File received successfully, session closed, arg contains closed session handle
TFTPDLL_EV_FILE_TX_COMPLETE	File sent successfully, session closed, arg contains closed session handle
TFTPDLL_EV_ERROR	Session aborted due to ERROR packet received from remote peer, arg contains closed session handle
TFTPDLL_EV_TIMEOUT	Session closed due to timeout, arg contains closed session handle
TFTPDLL_EV_FWR_ERROR	Session closed due to local file write error, arg contains closed session handle
TFTPDLL_EV_WRITEREQ_REJ_EVAL	file write request rejected - use count exceeded in evaluation version
TFTPDLL_EV_READREQ_REJ_EVAL	file read request rejected - use count exceeded in evaluation version

Return values:

HS_TFTP_RC_OK	Success and evt and arg contain further information about the event that has been read
HS_TFTP_RC_INVALID_PAR	Invalid parameters

Sample usage:

```
using HsTftpDllCs;

private void timer1_Tick(object sender, EventArgs e)
{
    int rc = HS_TFTP_RC_FAIL;
    String errorString;
```

```

int evt = TFTPDLL_NO_EVENTS;
int arg = 0;
StringBuilder remote_ip = new StringBuilder(16);
StringBuilder filename = new StringBuilder(256);
String str;

rc = HsTftp.HsTftpDllReadEvents(ref evt, ref arg);
if (rc != HS_TFTP_RC_OK)
{
    errorString = String.Format("ReadEvents failed RC {0}", rc);
    write_event(errorString);
    return;
}

switch (evt)
{
    case TFTPDLL_NO_EVENTS:
        break;

    case TFTPDLL_EV_FILE_RX_START:
        errorString = String.Format("File reception started (session handle: {0})", arg);
        write_event(errorString);

        Form1_populate_session(arg);
        break;

    case TFTPDLL_EV_FILE_RX_COMPLETE:
        errorString = String.Format("File reception complete (session handle: {0})", arg);
        write_event(errorString);

        str = String.Format("{0}", arg);
        listView1.Items.RemoveByKey(str);
        break;

    case TFTPDLL_EV_FILE_TX_START:
        errorString = String.Format("File transmission started (session handle: {0})", arg);
        write_event(errorString);

        Form1_populate_session(arg);
        break;

    case TFTPDLL_EV_FILE_TX_COMPLETE:
        errorString = String.Format("File transmission complete (session handle: {0})", arg);
        write_event(errorString);

        str = String.Format("{0}", arg);
        listView1.Items.RemoveByKey(str);
        break;

    case TFTPDLL_EV_ERROR:
        errorString = String.Format("Transfer Failed (error received) (session handle: {0})", arg);
        write_event(errorString);

        str = String.Format("{0}", arg);
        listView1.Items.RemoveByKey(str);
        break;
}

```

```

case TFTPDLL_EV_TIMEOUT:
    errorString = String.Format("Transfer Failed - Timeout (session handle: {0})", arg);
    write_event(errorString);

    str = String.Format("{0}", arg);
    listView1.Items.RemoveByKey(str);
    break;

case TFTPDLL_EV_FWR_ERROR:
    errorString = String.Format("Transfer Failed - file write error (session handle: {0})", arg);
    write_event(errorString);

    str = String.Format("{0}", arg);
    listView1.Items.RemoveByKey(str);
    break;
}
}

```

2.5 HsTftpDllGetSessionInfo

Declaration:

```

[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllGetSessionInfo(
    int session_handle,
    ref int type,
    ref int operation,
    StringBuilder filename,
    StringBuilder remote_ip,
    ref long block_nr,
    ref long bytes_nr
);

```

Summary:

Gets session information given session handle: session type (Client or Server), operation (Send or Receive), filename, remote IP address, block numbers transferred, number of bytes transferred.

Parameters:

session_handle – TFTP session handle

type – returns session type:

TFTPDLL_SESSION_SERVER (0x1) – Server session;

TFTPDLL_SESSION_CLIENT (0x2) – Client session

operation – returns transfer direction of the session:

TFTPDLL_SESSION_TX (0x1) – file send operation

TFTPDLL_SESSION_RX (0x2) – file receive operation

filename – returns filename of the file being sent or received

remote_ip – returns remote IP address string

block_nr – returns number of blocks transferred

bytes_nr – returns number of bytes transferred

Return values:

HS_TFTP_RC_OK	Success
HS_TFTP_RC_NOT_INIT	TFTP .NET library not initialised
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FAIL	Session handle is invalid – session not allocated

Sample usage:

```
using HsTftpDllCs;

private void Form1_populate_session(int session_handle)
{
    int rc;
    String str;
    String str1;
    String errorString;
    int type = 0;
    int operation = 0;
    StringBuilder remote_ip = new StringBuilder(16);
    StringBuilder filename = new StringBuilder(256);
    long block_nr = 0;
    long bytes_nr = 0;
    int index = 0;

    rc = HsTftp.HsTftpDllGetSessionInfo(session_handle, ref type, ref operation, filename,
        remote_ip, ref block_nr, ref bytes_nr);

    if (rc != HS_TFTP_RC_OK)
    {
        errorString = String.Format("failed to get session info RC {0}", rc);
        write_event(errorString);
        return;
    }

    str = String.Format("{0}", session_handle);
    listView1.Items.Add(str, 0);

    index = listView1.Items.IndexOfKey(str);

    str1 = String.Format("{0}", (type == TFTPDLL_SESSION_SERVER) ? "Srv" : "Cli");
    listView1.Items[index].SubItems.Add(str1);

    str1 = String.Format("{0}", (operation == TFTPDLL_SESSION_TX) ? "Send" : "Recv");
    listView1.Items[index].SubItems.Add(str1);

    str1 = String.Format("{0}", filename);
    listView1.Items[index].SubItems.Add(str1);

    str1 = String.Format("{0}", remote_ip);
    listView1.Items[index].SubItems.Add(str1);

    str1 = String.Format("{0}", block_nr);
    listView1.Items[index].SubItems.Add(str1);
}
```

```

    str1 = String.Format("{0}", bytes_nr);
    listView1.Items[index].SubItems.Add(str1);
}

```

2.6 HsTftpDllGetSessionStats

Declaration:

```

[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllGetSessionStats(
    int session_handle,
    ref long block_nr,
    ref long bytes_nr
);

```

Summary:

Gets session statistics: number of blocks and number of bytes transferred so far

Parameters:

session_handle – TFTP session handle

block_nr – returns number of blocks transferred

bytes_nr – returns number of bytes transferred

Return values:

HS_TFTP_RC_OK	Success
HS_TFTP_RC_NOT_INIT	TFTP .NET library not initialised
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FAIL	Session handle is invalid – session not allocated

Sample usage:

```

using HsTftpDllCs;

private void timer2_Tick(object sender, EventArgs e)
{
    int rc;
    long block_nr = 0;
    long bytes_nr = 0;
    int i;
    String str;
    int session_handle = 0;

    if (listView1.Items.Count == 0)
        return;

    for (i = 0; i < listView1.Items.Count; i++)
    {
        str = listView1.Items[i].SubItems[0].Text;
        session_handle = Convert.ToInt16(str);

        rc = HsTftp.HsTftpDllGetSessionStats(session_handle, ref block_nr, ref bytes_nr);
    }
}

```

```

    if (rc == HS_TFTP_RC_OK)
    {
        str = String.Format("{0}", block_nr);
        listView1.Items[i].SubItems[5].Text = str;

        str = String.Format("{0}", bytes_nr);
        listView1.Items[i].SubItems[6].Text = str;
    }
}
}

```

2.7 HsTftpDllClientSendFile

Declaration:

```

[DllImport("HsTftpDll.Dll")]
public static extern int HsTftpDllClientSendFile(
    String ip_address,      // remote server IP address
    int port,              // remote port
    String filename,       // filename to send
    int user_handle,       // caller handle
    ref int session_handle // session handle
);

```

Summary:

Initiates file send operation in client mode

Parameters:

ip_address – remote server ip address

port – remote TFTP server port

filename – filename of file to send

user_handle – calling code handle

session_handle – returns TFTP session handle

Return values:

HS_TFTP_RC_OK	File send operation successfully started
HS_TFTP_RC_EVAL_USE_EXCEEDED	In evaluation version only, if this code is returned, it means that the maximum number of file transfers allowed in evaluation version has been exceeded
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FOPEN	File open failure
HS_TFTP_RC_NO_FREE_CTX	No free TFTP session contexts remaining, cannot allocate new session
HS_TFTP_RC_UDP SOCK_OPEN	Socket open failure
HS_TFTP_RC_UDP SOCK_SEND	Socket send failure

Sample usage:

```
using HsTftpDllCs;
```

```
String str;
int rc;

rc = HsTftp.HsTftpDllClientSendFile(textBoxIP.Text, port, textBox1.Text, 5,
                                     ref session_handle);

str = String.Format("Started client file send: {0} Rc: ({1})",
                   (rc == HS_TFTP_RC_OK) ? "Ok" : "Fail", rc);

write_event(str);
```

2.8 HsTftpDllClientReceiveFile

Declaration:

```
[DllImport("HsTftpDll.Dll")]
public static extern int HsTftpDllClientReceiveFile(
    String ip_address,      // remote server IP address
    int port,              // remote port
    String filename,       // filename to receive
    int user_handle,       // caller handle
    ref int session_handle // session handle
);
```

Summary:

Initiates file send operation in client mode

Parameters:

ip_address – remote server ip address

port – remote TFTP server port

filename – filename of file to send

user_handle – calling code handle

session_handle – returns TFTP session handle

Return values:

HS_TFTP_RC_OK	File send operation successfully started
HS_TFTP_RC_EVAL_USE_EXCEEDED	In evaluation version only, if this code is returned, it means that the maximum number of file transfers allowed in evaluation version has been exceeded
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FOPEN	File open failure
HS_TFTP_RC_NO_FREE_CTX	No free TFTP session contexts remaining, cannot allocate new session
HS_TFTP_RC_UDP SOCK_OPEN	Socket open failure
HS_TFTP_RC_UDP SOCK_SEND	Socket send failure

Sample usage:

```
using HsTftpDllCs;
```

```
String str;
int rc;
```

```
rc = HsTftp.HsTftpDllClientReceiveFile(textBoxIP.Text, port, textBox1.Text, 5, ref session_handle);
str = String.Format("Started client file receive: {0} Rc: ({1})",
    (rc == HS_TFTP_RC_OK) ? "Ok" : "Fail", rc);
write_event(str);
```

2.9 HsTftpDllGetIPAddrFirst

Declaration:

```
[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllGetIPAddrFirst(
    StringBuilder ip_str_buf,
    ref int num_ip
);
```

Summary:

Gets first detected ip address string from the list of all detected local IP addresses. IP address string is in dotted IP format x.x.x.x of maximum 16 bytes including zero terminating character. This function together with HsTftpDllGetIPAddrNext may be used to present a list of local IP addresses to a user for selecting which local IP address to use. The actual IP address to use is set with function HsTftpDllSetIpIndex. If HsTftpDll was not initialized it gets initialized.

Parameters:

ip_str_buf – local ip address string in dotted format (x.x.x.x), zero terminated of type StringBuilder. Length of buffer required is 16 bytes

num_ip – This integer variable receives number of detected local IP addresses on exit from the function

Return values:

HS_TFTP_RC_OK	Success, ip address retrieved
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FAIL	Socket layer failed to retrieve local IP address

Sample usage:

```
using HsTftpDllCs;

StringBuilder ipaddrstr = new StringBuilder(16);
int num_ip = 0;
int index;

index = 0;
rc = HsTftp.HsTftpDllGetIPAddrFirst(ipaddrstr, ref num_ip);
if (rc != HS_TFTP_RC_OK)
{
    write_event("No Local Ip Addresses Detected");
    return;
}

// at least first address detected
```

```

AddIpAddressEntry(index, ipaddrstr);

listViewLoclp.Items[index].Focused = true;
listViewLoclp.Items[index].Selected = true;

while (rc == HS_TFTP_RC_OK)
{
    rc = HsTftp.HsTftpDllGetIPAddrNext(ipaddrstr);
    if (rc == HS_TFTP_RC_OK)
    {
        index++;
        AddIpAddressEntry(index, ipaddrstr);
    }
}

number_of_local_ip_addresses = (index+1);

```

2.10 HsTftpDllGetIPAddrNext

Declaration:

```

[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllGetIPAddrNext(
    StringBuilder ip_str_buf
);

```

Summary:

Gets next detected ip address string from the list of all detected local IP addresses. IP address string is in dotted IP format x.x.x.x of maximum 16 bytes including zero terminating character. This function together with HsTftpDllGetIPAddrFirst may be used to present a list of local IP addresses to a user for selecting which local IP address to use. The actual IP address to use is set with function HsTftpDllSetIpIndex. If HsTftpDll was not initialized it gets initialized.

Parameters:

ip_str_buf – local ip address string in dotted format (x.x.x.x), zero terminated of type StringBuilder. Length of buffer required is 16 bytes

Return values:

HS_TFTP_RC_OK	Success, ip address retrieved
HS_TFTP_RC_INVALID_PAR	Invalid parameters
HS_TFTP_RC_FAIL	Socket layer failed to retrieve local IP address
HS_TFTP_RC_NOT_INIT	HdTftpDll Library not initialised

Sample usage:

```

using HsTftpDllCs;

StringBuilder ipaddrstr = new StringBuilder(16);
int num_ip = 0;
int index;

index = 0;
rc = HsTftp.HsTftpDllGetIPAddrFirst(ipaddrstr, ref num_ip);

```

```

if (rc != HS_TFTP_RC_OK)
{
    write_event("No Local Ip Addresses Detected");
    return;
}

// at least first address detected
AddIpAddressEntry(index, ipaddrstr);

listViewLoclp.Items[index].Focused = true;
listViewLoclp.Items[index].Selected = true;

while (rc == HS_TFTP_RC_OK)
{
    rc = HsTftp.HsTftpDllGetIPAddrNext(ipaddrstr);
    if (rc == HS_TFTP_RC_OK)
    {
        index++;
        AddIpAddressEntry(index, ipaddrstr);
    }
}

number_of_local_ip_addresses = (index+1);

```

2.11 HsTftpDllSetIpIndex

Declaration:

```

[DllImport("HsTftpDll.dll")]
public static extern int HsTftpDllSetIpIndex (
    int index
);

```

Summary:

Sets local IP address index to use for server and client operations. The index is zero based. If there is more than one IP interface in the system, the HsTftpDll shall detect and store a list of local IP addresses. With this function it is possible to tell HsTftpDll which local IP address to use for TFTP server and client. The list of detected local IP addresses can be retrieved with functions HsTftpDllGetIPAddrFirst and HsTftpDllGetIPAddrNext. By default (if this function is not called) first detected IP address shall be used.

Parameters:

index – integer index of local IP address to use (zero based)

Return values:

HS_TFTP_RC_OK	Success, ip address index set
---------------	-------------------------------

2.12 Return Codes, Events and Other Definitions

2.12.1 HS TFTP .NET return codes

Name	Value	Comment
HS_TFTP_RC_FAIL	0	Failure
HS_TFTP_RC_OK	1	Success
HS_TFTP_RC_INVALID_PAR	2	Invalid parameters
HS_TFTP_RC_NO_MEM	3	No free memory
HS_TFTP SOCK_STARTUP_FAIL	4	Socket layer startup failed
HS_TFTP_RC_INIT	5	TFTP library already initialised
HS_TFTP_RC_NOT_INIT	6	TFTP library not initialised
HS_TFTP_RC_NO_FREE_CTX	7	No free TFTP contexts
HS_TFTP_RC_UDP SOCK_OPEN	8	UDP socket open failed
HS_TFTP_RC_UDP SOCK_SEND	9	UDP socket send failed
HS_TFTP_RC_FOPEN	10	File open error
HS_TFTP_RC_SETDIR	11	Cannot set directory
HS_TFTP_RC_EVAL_USE_EXCEEDED	12	Evaluation version use count exceeded

2.12.2 HS TFTP .NET event notifications

Name	Value	Comment
TFTPDLL_EV_FILE_RX_START	1	File reception started
TFTPDLL_EV_FILE_RX_COMPLETE	2	File reception complete, file is stored to disk
TFTPDLL_EV_FILE_TX_START	3	File transmission started
TFTPDLL_EV_FILE_TX_COMPLETE	4	File transmission complete
TFTPDLL_EV_ERROR	5	Session aborted due to ERROR packet received from remote peer
TFTPDLL_EV_TIMEOUT	6	Session closed due to timeout
TFTPDLL_EV_FWR_ERROR	7	Session closed due to local file write error
TFTPDLL_NO_EVENTS	8	Event queue empty, there are no new events
TFTPDLL_EV_WRITEREQ_REJ_EVAL	9	File write request rejected - use count exceeded in evaluation version
TFTPDLL_EV_READREQ_REJ_EVAL	10	File read request rejected - use count exceeded in evaluation version

2.12.3 Other Definitions

Name	Value	Comment
TFTPDLL_SESSION_SERVER	1	Server session
TFTPDLL_SESSION_CLIENT	2	Client session
TFTPDLL_SESSION_TX	1	Send operation
TFTPDLL_SESSION_RX	2	Receive operation