TFTP User Guide

Version 1.80

For use with Trivial File Transfer Protocol (TFTP) module versions 3.04 and above

Exported on 10/05/2018

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

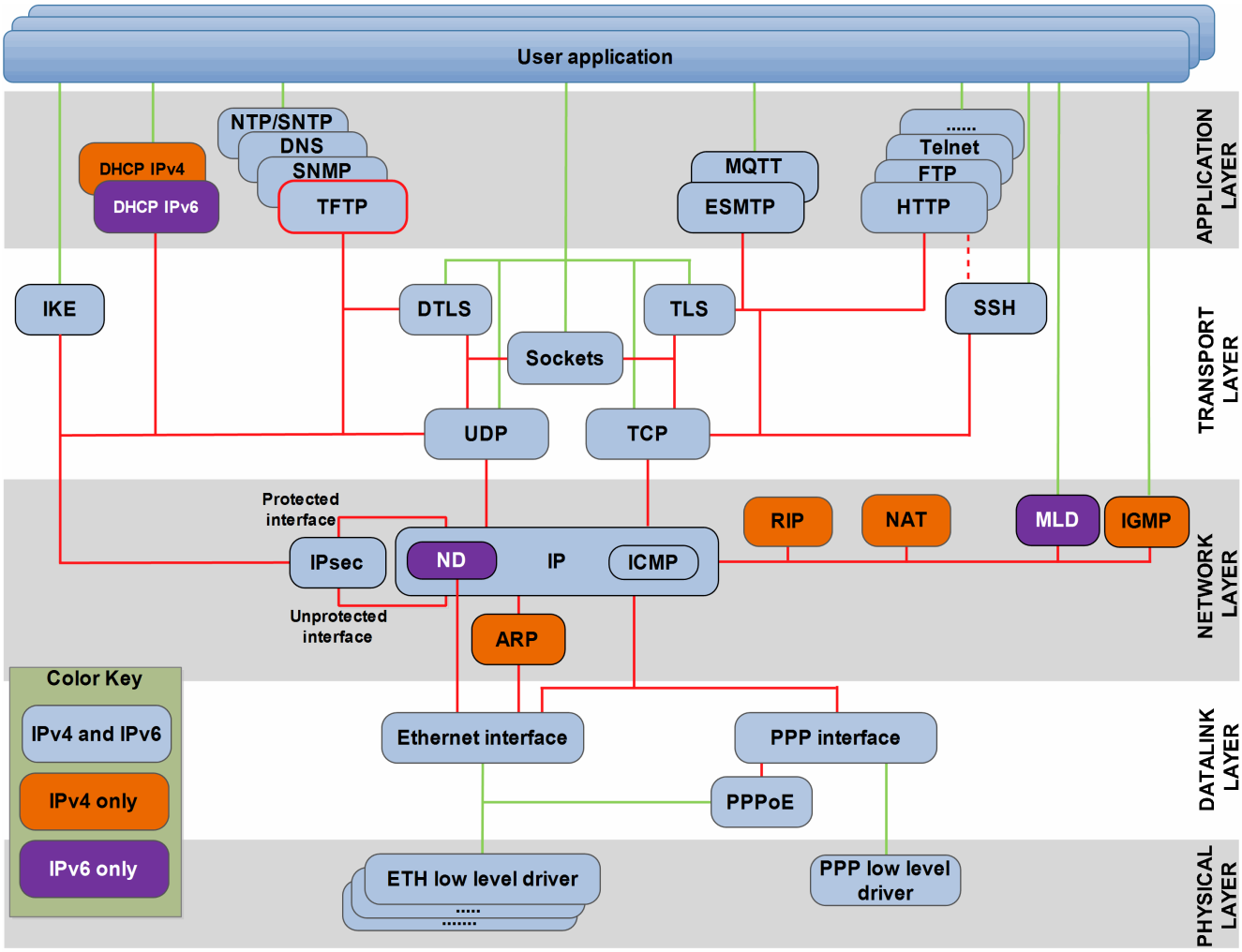
- Introduction – describes the main elements of the module. This section includes a diagram showing the position of this module within HCC's TCP/IP stack.
- Feature Check – summarizes the main features of the module as bullet points.
- Overview of Operation - presents time sequence diagrams that show how the module operates.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

Note: To download this manual as a PDF, see TCP/IP PDFs.

1.1 Introduction

This guide is for those who want to implement a Trivial File Transfer Protocol (TFTP) server and/or client as part of HCC Embedded's TCP/IP stack.

The TFTP module is part of HCC's MISRA-compliant TCP/IP stack, as shown below, and is designed specifically for use with it. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



TFTP is a standard network protocol used to transfer files from one host to another over a TCP-based network. This module implements TFTP server and client functions that are used to transfer files over the UDP protocol. You can use the module in two ways:

- To set up a TFTP server. A remote device can request (get) files from this server and also transfer (put) files to it.
- As a client function that an app on a device can call to transfer files to/from a remote TFTP server.

Any file system or RAM system can be attached to the TFTP server. To do this, you just need to implement the simple callback functions described in Callback Functions.

The HCC TFTP module is designed specifically for use with the HCC TCP/IP stack. It is fast and simple. Once the system is initialized, there are just three types of Application Programming Interface (API) function:

- Start and stop the TFTP module.
- Start and stop the TFTP server.
- Start the TFTP client.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Complies with HCC's MISRA-compliant TCP/IP stack.
- Fully compatible with the HCC IPv4 and IPv6 stacks.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to RFC 1350 (TFTP Protocol Revision 2).
- Has an option allowing you to enable/disable the TFTP client.
- Can interface to any file system by using user-provided callback functions.
- Can interface to RAM by using user-provided callback functions.
- Number of simultaneous server connections is configurable.

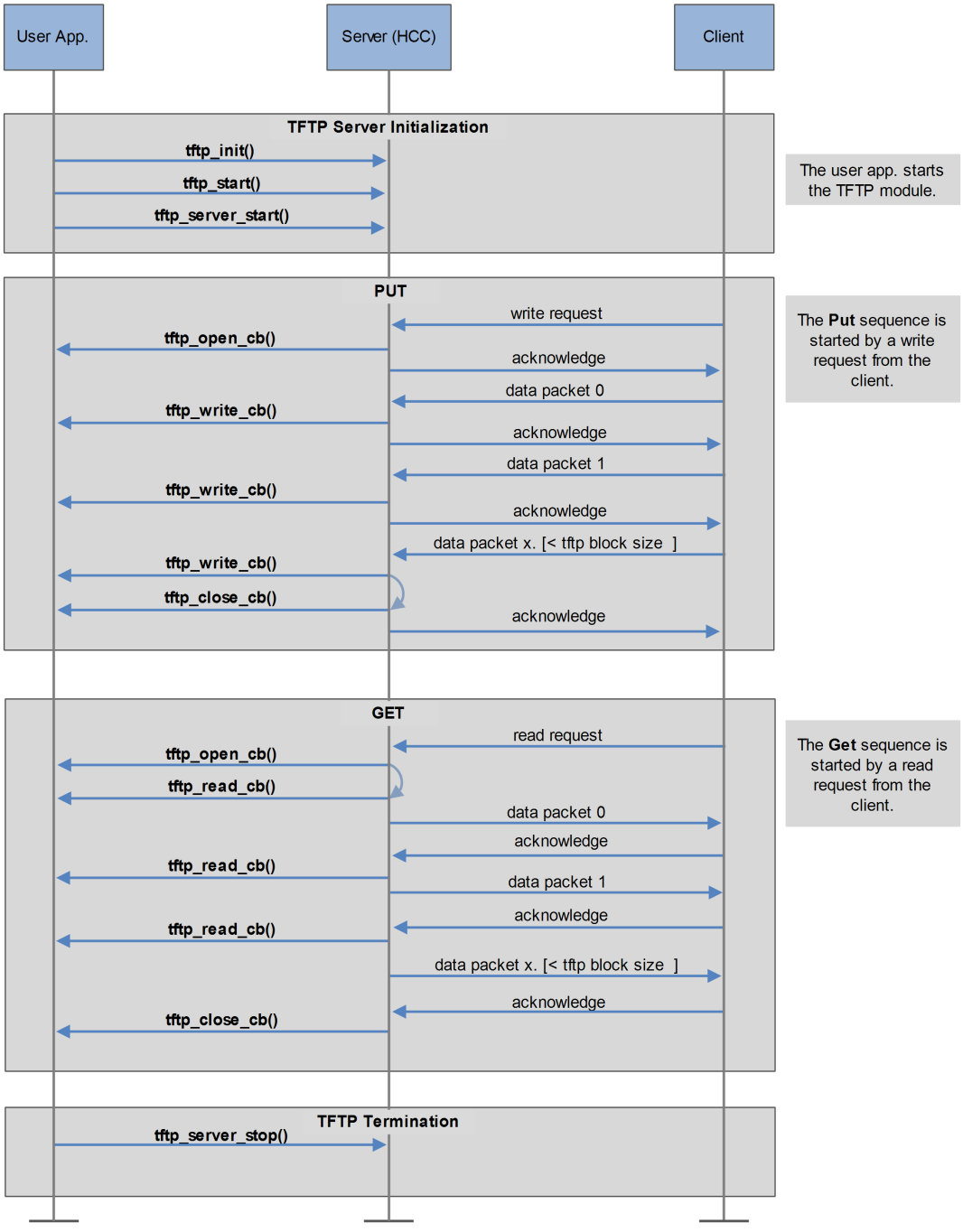
1.3 Overview of Operation

This section contains a set of time sequence diagrams showing how the TFTP protocol works and how it interacts with a supporting file system by using the callback functions provided.

Normal TFTP Server Operation

This diagram shows:

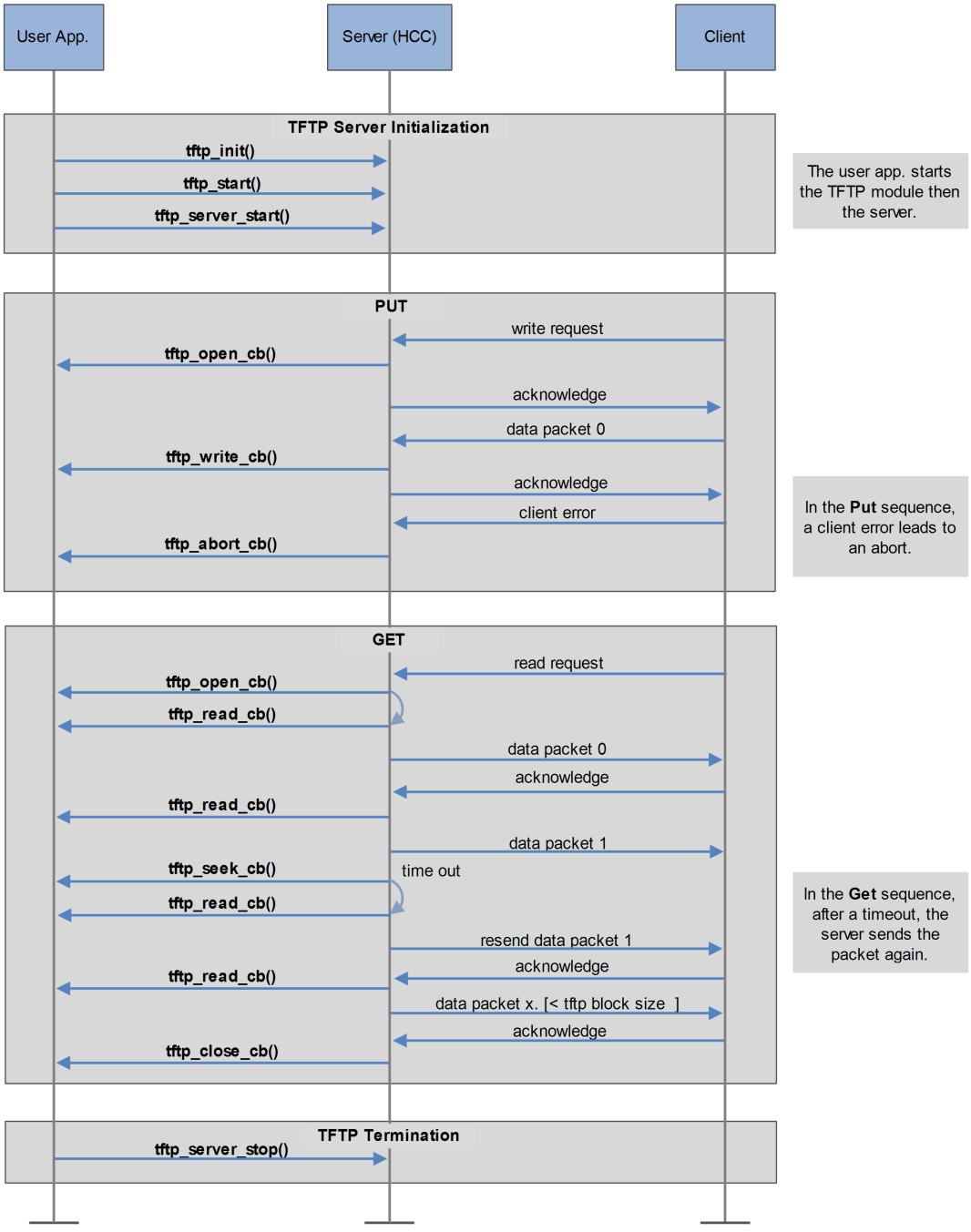
1. TFTP server initialization.
2. A TFTP PUT sequence where the client uploads a file.
3. A TFTP GET sequence where the client downloads a file.
4. TFTP server termination.



TFTP Server Operation with Errors

This diagram shows:

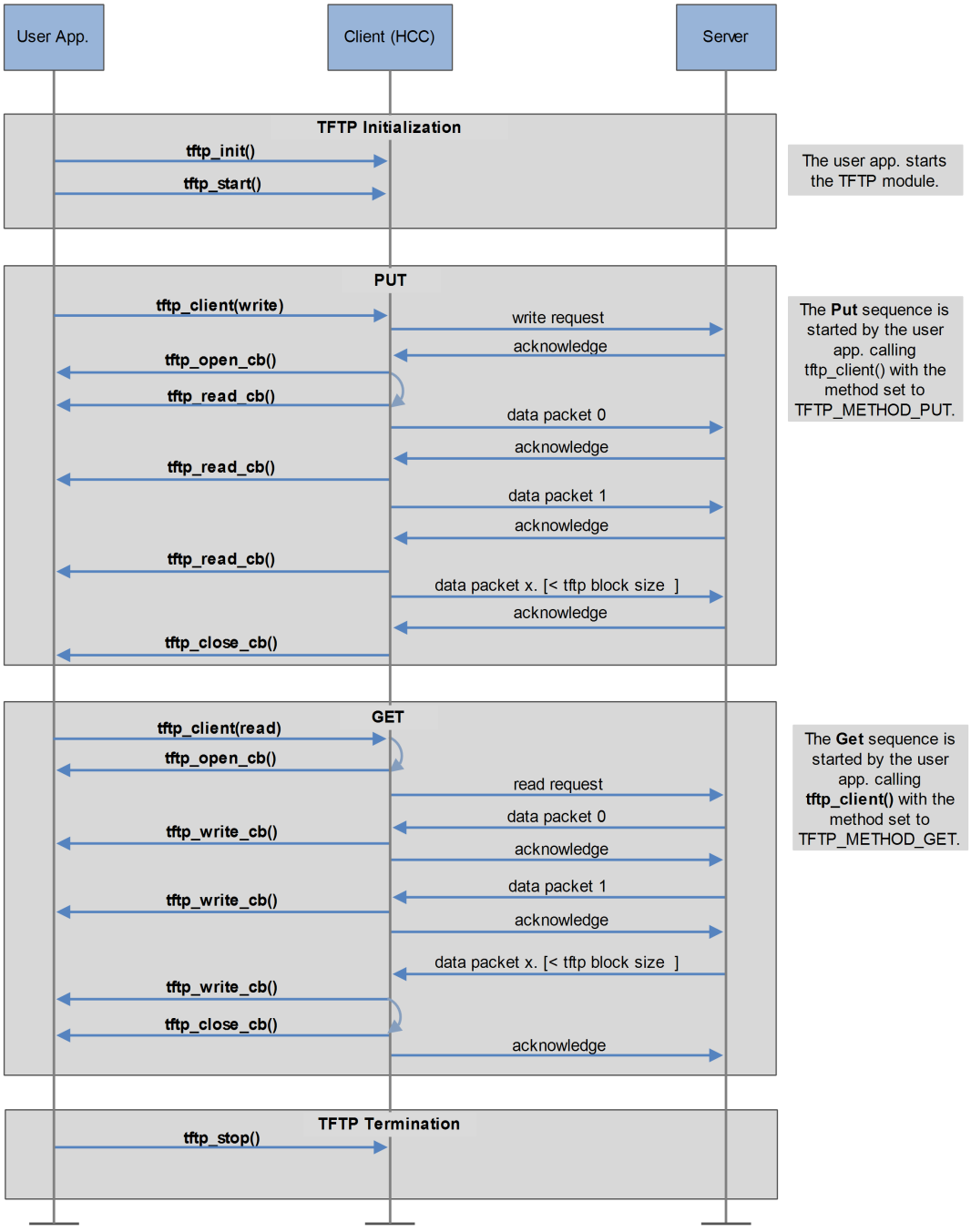
1. TFTP server initialization.
2. A TFTP PUT sequence where a client tries to upload a file but generates an error during the transfer.
3. A TFTP GET sequence where a client downloads a file but there is a timeout during the transfer.
4. TFTP server termination.



Normal TFTP Client Operation

This diagram shows:

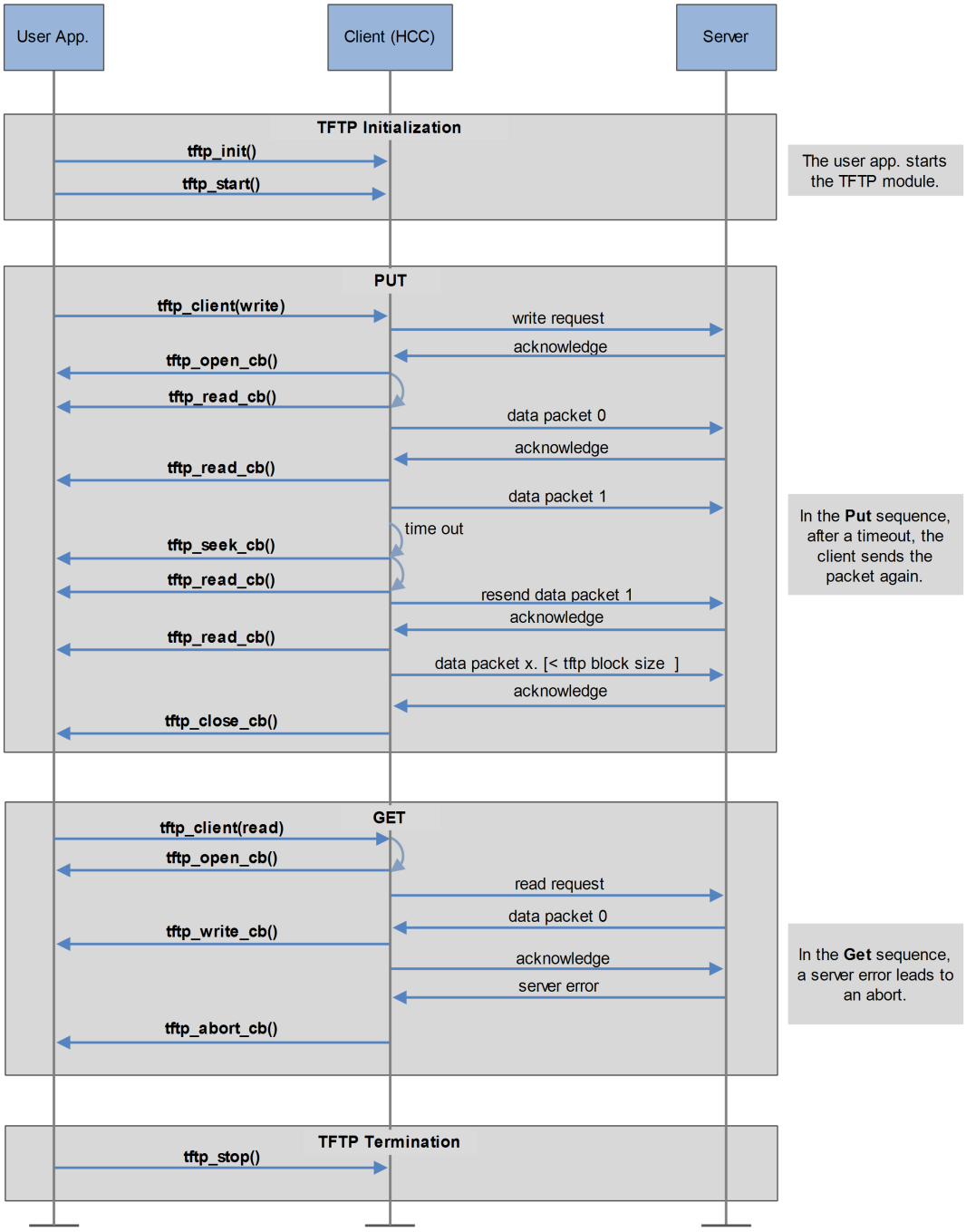
1. TFTP client initialization.
2. A TFTP PUT sequence where the client uploads a file to the server.
3. A TFTP GET sequence where the client downloads a file from the server.
4. TFTP client termination.



TFTP Client Operation with Errors

This diagram shows:

1. TFTP client initialization.
2. A TFTP PUT sequence where the client uploads a file. A timeout occurs during the transfer but this is recovered with the help of a seek callback.
3. A TFTP GET sequence where the client downloads a file but the server generates an error during the transfer.
4. TFTP client termination.



1.4 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
ip_app_tftp	The TFTP package described in this manual.
ip_app_tftp_demo	The TFTP demo package. This has example code showing how to use the required Callback Functions to interface to both file systems and RAM buffers.
mip_base	The TCP/IP Dual Stack base package.
mip_udp	The UDP package.

Documents

For an overview of the HCC TCP/IP stack software, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC TCP/IP Dual Stack System User Guide

This is the core document that describes the complete TCP/IP stack. It covers both IPv4 and IPv6 systems.

HCC TFTP User Guide

This is this document.

1.5 Change History

This section describes past changes to this manual.

- To download this manual or a PDF describing an earlier software version, see TCP/IP PDFs.
- For the history of changes made to the package code itself, see History: ip_app_tftp and History: ip_app_tftp_demo.

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

Manual version	Date	Software version	Reason for change
1.80	2018-10-05	3.05	Added notes on three *Server and Client Functions* pages. Reordered columns in *t_tftp_cb_dsc.*
1.70	2018-06-06	3.05	Reorganized *Overview of Operation* section.
1.60	2017-06-20	3.04	New *Change History* format.
1.50	2017-03-28	3.03	Updated network diagram.
1.40	2017-01-16	3.03	Updated network diagram, *Feature Check*.
1.30	2017-01-16	3.02	Updated network diagram.
1.20	2015-10-06	2.03	Added software change history to manual.
1.10	2015-09-04	2.03	Reorganized *System Overview* section.
1.00	2015-07-07	2.03	First online version.

2 Source File List

The following sections describe all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the HCC Source Tree Guide. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

Note: Do not modify any files except the configuration files.

2.1 API Header File

The file **src/api/api_ip_app_tftp.h** is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

2.2 Configuration Files

These files are in the directory **src/config**. Configure them as required:

File	Description
config_ip_app_tftp.h	The configurable TFTP options. For details, see config_ip_app_tftp.h.
config_ip_app_tftp.c	Sets the error message sent to the remote node by a callback function. For details, see config_ip_app_tftp.c.

2.3 Source Code

The file **src/ip/apps/tftp/tftp.c** is the main TFTP source code file. **This file should only be modified by HCC.**

2.4 Version

The file **src/version/ver_ip_app_tftp.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

There are two configuration files, as described below.

3.1 config_ip_app_tftp.h

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

TFTP_TASK_STACK_SIZE

The TFTP task stack size. The default value is 4096.

> **Note:** All file system (callback) functions are called from the TFTP task context .

TFTP_MAX_CONNS

The maximum number of server/client connections that can be open simultanously. The default value is 4.

TFTP_CONN_PORT_START

The first connection port. The ports are used in rotation for every new connection. The default value is 12000.

TFTP_CONN_PORT_COUNT

The number of ports. The default value is 500.

TFTP_SERVER_ENABLE

Keep this at the default value of 1 to enable server functionality.

TFTP_SERVER_PORT

The server port number. The default value is 69.

TFTP_CLIENT_ENABLE

Keep this at the default value of 1 to enable client operation. To disable it set it to 0.

TFTP_FNAME_LEN

The maximum file name length. The default value is 40.

TFTP_TIMEOUT

The retry timeout used when the remote node does not respond, in seconds. The default value is 5.

3.2 config_ip_app_tftp.c

Use the file **src/config/config_ip_app_tftp.c** to configure the error message sent to the remote node if an error is returned by a callback function.

The default is as follows:

```
const char_t * g_tftp_errc_msg[TFTP_ERRC_COUNT] =
{
    "Undefined"
    , "File not found"
    , "Access violation"
    , "Disk full"
    , "Illegal command"
    , "Unknown transfer ID"
    , "File exists"
    , "No such user"
};
```

4 Application Programming Interface

This section documents the Application Programming Interface (API). It includes all the functions that are available to an application program.

4.1 Module Management

The functions are the following:

Function	Description
tftp_init()	Initializes the module and allocates the required resources.
tftp_start()	Starts the module.
tftp_stop()	Stops the module.
tftp_delete()	Deletes the module and releases the resources it used.

tftp_init

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

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

Format

```
t_tftp_ret tftp_init ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	Operation failed.

tftp_start

Use this function to start the TFTP module.

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

Format

```
t_tftp_ret tftp_start ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	Operation failed.
TFTP_ERR_PROTO	Protocol error (server port failed to open).
TFTP_ERR_STARTED	The module has already been started.

tftp_stop

Use this function to stop the TFTP module.

Format

```
t_tftp_ret tftp_stop ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	Operation failed.
TFTP_ERR_PROTO	Protocol error (server port failed to close).
TFTP_ERR_STOPPED	The module has already been stopped.

tftp_delete

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

Format

```
t_tftp_ret tftp_delete ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	Operation failed.

4.2 Server and Client Functions

These functions control the server and client.

Function	Description
tftp_server_start()	Starts the TFTP server.
tftp_server_stop()	Stops the TFTP server.
tftp_client()	Executes a client GET/PUT request.

tftp_server_start

Use this function to start the TFTP server.

> **Note:** This is only available if configuration option TFTP_SERVER_ENABLE is set.

Format

```
t_tftp_ret tftp_server_start ( const t_tftp_cb_dsc * const p_cb_dsc )
```

Arguments

Arguments	Description	Type
p_cb_dsc	A pointer to the callback function descriptor.	t_tftp_cb_dsc *

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERR_INVALID_PARAM	Either the pointer to the callback descriptor is invalid, or a callback itself is invalid.
TFTP_ERR_PROTO	Protocol error (server port failed to open).
TFTP_ERR_STARTED	The server has already been started.

tftp_server_stop

Use this function to stop the TFTP server.

> **Note:** This is only available if configuration option TFTP_SERVER_ENABLE is set.

Format

```
t_tftp_ret tftp_server_stop ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERR_PROTO	Protocol error (server port failed to close).
TFTP_ERR_STOPPED	The server has already been stopped.

tftp_client

Use this function to execute a client GET/PUT request.

> **Note:** This is only available if configuration option TFTP_CLIENT_ENABLE is set.

This is a non-blocking function. To follow the state of the request, use the callback functions.

Format

```
t_tftp_ret tftp_client (
    const t_tftp_method           method,
    t_ip_addr * const             p_ip_addr,
    const char_t * const          p_filename,
    const t_tftp_cb_dsc * const   p_cb_dsc )
```

Arguments

Arguments	Description	Type
method	TFTP_METHOD_GET or TFTP_METHOD_PUT.	t_tftp_method
p_ip_addr	A pointer to the remote IP address.	t_ip_addr *
p_filename	A pointer to the file name.	char_t *
p_cb_dsc	A pointer to the callback descriptor.	t_tftp_cb_dsc *

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERR_INVALID_CB	A callback is invalid.
TFTP_ERR_INVALID_PARAM	Invalid IP address.
TFTP_ERR_INVALID_SIZE	The filename length exceeds TFTP_FNAME_LEN.
TFTP_RETRY	Retry later (another request is in progress).

4.3 Error Codes

If a function executes successfully, it returns with TFTP_SUCCESS. The following table shows the meaning of the TFTP return codes.

Note: Also check other error code values used in the TCP base system; see the *HCC TCP/IP Dual Stack System User Guide*.

Return Value	Value	Description
TFTP_SUCCESS	0	Successful execution.
TFTP_RETRY	1	Retry later (another request is in progress).
TFTP_ERR_NO_MORE_ENTRY	2	There are no more entries.
TFTP_INVALID_PARAM_ERR	3	A parameter is invalid.
TFTP_ERR_INVALID_CB	4	Invalid parameter.
TFTP_ERR_INVALID_SIZE	5	Invalid size.
TFTP_ERR_PROTO	6	Protocol error.
TFTP_ERR_CB	7	Callback returned error.
TFTP_ERR_TIMEOUT	8	Timeout error.
TFTP_ERR_STARTED	9	The module has already been started.
TFTP_ERR_STOPPED	10	The module has already been stopped.
TFTP_ERR_REMOTE	11	Error occurred on the remote node.
TFTP_ERROR	12	General error.

4.4 Types and Definitions

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

t_tftp_method

The *t_tftp_method* typedef defines two transfer methods, as follows:

Name	Description
TFTP_METHOD_GET	Get: request a file from a server.
TFTP_METHOD_PUT	Put: send a file to a server.

t_tftp_mode

The file open modes are defined in *t_tftp_mode*, as follows:

Name	Description
TFTP_MODE_READ	Text read.
TFTP_MODE_WRITE	Text write.

t_tftp_cb_dsc

The *t_tftp_cb_dsc* structure is the callback function descriptor:

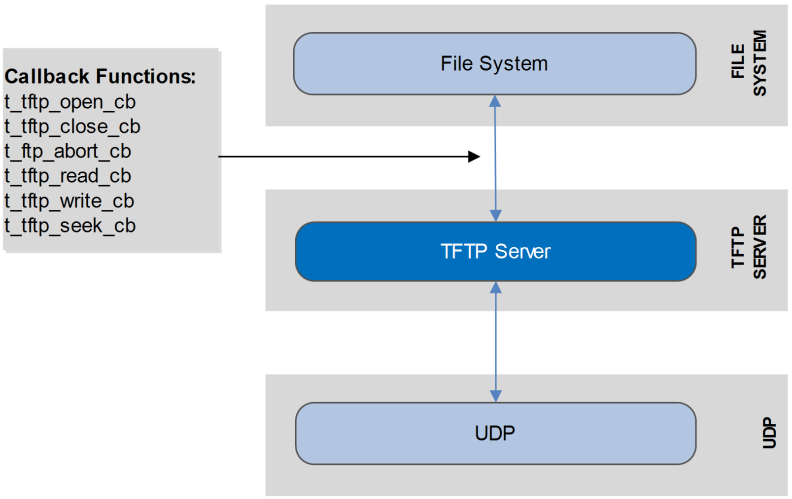
Element	Type	Description
tcd_open_cb	t_tftp_open_cb	The open callback.
tcd_close_cb	t_tftp_close_cb	The close callback.
tcd_abort_cb	t_tftp_abort_cb	The abort callback.
tcd_read_cb	t_tftp_read_cb	The read callback.
tcd_write_cb	t_tftp_write_cb	The write callback.
tcd_seek_cb	t_tftp_seek_cb	The seek callback.

5 Integration

This section describes all aspects of the module that require integration with your target project. This includes porting and configuration of external resources.

5.1 Interfacing to a File System

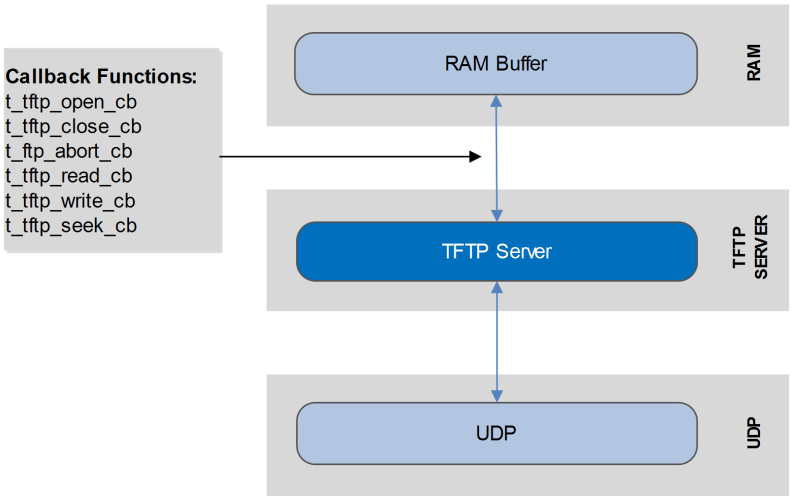
Interfacing to a file system allows you to transfer data between (both to and from) a file system and a remote TFTP server client. To do this, you use the callback functions provided, as shown below.



The demo package file **tftp_user_fs.c** has examples of code for each callback function that you can use to interface to the file system.

5.2 Interfacing to RAM

Interfacing to a RAM system allows you to transfer data between (both to and from) a RAM buffer and a remote TFTP server client. To do this, you use the callback functions provided, as shown below.



This allows you to transfer data between RAM and a remote system that considers that area of RAM to be a file. There are many potential applications for this type of spoofed access.

The demo package file **tftp_user_buf.c** has examples of code for each callback function that you can use to interface to RAM.

5.3 The TFTP Demo Package

The TFTP demo package is named **ip_app_tftp_demo**.

The three files described below are located in the directory **psp\board\demo**.

tftp_user.h

This header file defines the four TFTP user module functions: **tftp_user_init()**, **tftp_user_start()**, **tftp_user_stop()** and **tftp_user_delete()**. These effectively make the four Module Management calls for you.

tftp_user_fs.c

This file shows how to interface to a file system. It does the following:

1. Defines the maximum number of files the module can use.
2. Declares function prototypes for each of the six callback functions.
3. Defines a callback descriptor and array of file handles as local variables.
4. Gives example code for each of the six callback functions.
5. Creates the TFTP user module functions listed above.

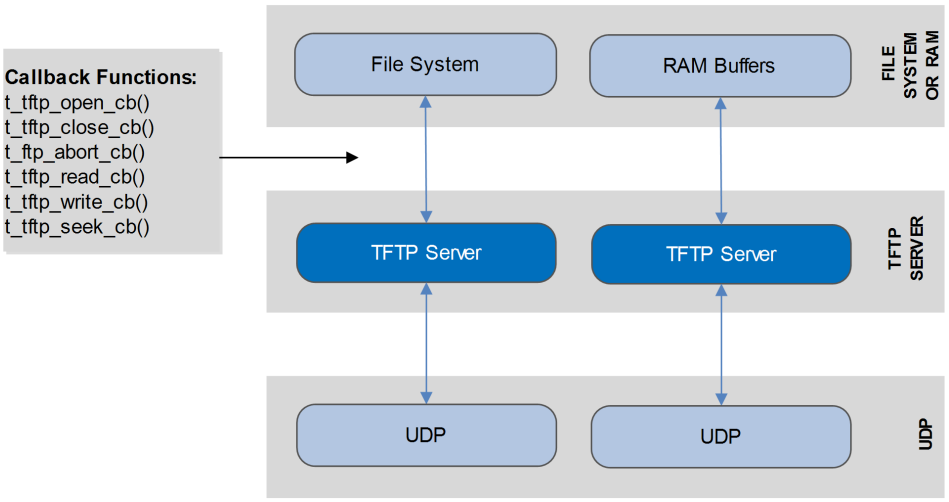
tftp_user_buf.c

This file shows how to interface to a RAM buffer. It does the following:

1. Defines the RAM buffer size.
2. Declares function prototypes for each of the six callback functions.
3. Defines a callback descriptor and RAM array as local variables.
4. Gives example code for each of the six callback functions.
5. Creates the TFTP user module functions listed above.

5.4 Callback Functions

The callback functions shown in the following diagram and described in this section provide a simple interface that allows you to attach any file system or RAM to the TFTP server:



All you have to do to connect to your file system or RAM is implement these callbacks. To help you with this, sample code for these callbacks is provided in the files of the TFTP demo package. There are examples for both file system and RAM. Use these as the basis for creating your own application.

Note:

- It is your responsibility to provide these callback functions, complying with the definitions described in this manual. You can implement as many of them as required.
- All callback functions are called from the same TFTP task context and are protected against concurrent calls. That is, one callback function cannot interrupt another.

The callbacks are the following:

Function	Description
t_tftp_open_cb()	Specifies the format of the callback function you may call to open a file for read or write.
t_tftp_close_cb()	Specifies the format of the callback function you may call to close a file.
t_tftp_abort_cb()	Specifies the format of the callback function you may call to abort a connection.
t_tftp_read_cb()	Specifies the format of the callback function you may call to read data.
t_tftp_write_cb()	Specifies the format of the callback function you may call to write data.
t_tftp_seek_cb()	Specifies the format of the callback function you may call to search in a file opened for read.

t_tftp_open_cb

The **t_tftp_open_cb** definition specifies the format of the callback function that may be called to open a file for read or write.

Note:

- If the file is opened for read, it must be in a mode that allows seeking later. This is required in case a packet needs to be retransmitted.
- The output handle cannot be TFTP_INVALID_CB_HDL.

Format

```
typedef t_tftp_errc ( * t_tftp_open_cb ) (
    char_t * const      p_filename,
    const t_tftp_mode   mode,
    uint32_t * const    p_hdl )
```

Arguments

Parameter	Description	Type
p_filename	A pointer to the name of the file to open.	char_t *
mode	The open mode.	t_tftp_mode
p_hdl	Where to write the internal handle.	uint32_t *

Return Codes

Code	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	Operation failed.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static t_tftp_errc tftp_open_cb ( char_t * const p_filename
                                , const t_tftp_mode mode
                                , uint32_t * const p_hdl )
{
    t_tftp_errc  ret_val;
    uint16_t     pos;

    f_enterFS();

    ret_val = TFTP_ERRC_FILENOTFND;

    for ( pos = 0u
        ; ( pos < MAX_FILES ) && ( g_file[pos] != (F_FILE *)NULL )
        ; pos++ )
    {
    }

    if ( pos < MAX_FILES )
    {
        if ( mode == TFTP_MODE_WRITE )
        {
            g_file[pos] = f_open( p_filename, "w" );
        }
        else
        {
            g_file[pos] = f_open( p_filename, "r" );
        }

        if ( g_file[pos] != NULL )
        {
            *p_hdl = pos;
            ret_val = TFTP_ERRC_SUCCESS;
        }
    }

    return ret_val;
} /* tftp_open_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static t_tftp_errc tftp_open_cb ( char_t * const p_filename
                                , const t_tftp_mode mode
                                , uint32_t * const p_hdl )
{
    t_tftp_errc  ret_val;

    HCC_UNUSED_ARG( p_filename );
    HCC_UNUSED_ARG( mode );

    ret_val = TFTP_ERRC_ACCESSVIO;
    if ( g_buf_used == FALSE )
    {
        if ( mode == TFTP_MODE_WRITE )
        {
            g_buf_size = 0u;
        }

        g_buf_pos = 0u;
        g_buf_used = TRUE;

        *p_hdl = 0u;

        ret_val = TFTP_ERRC_SUCCESS;
    }

    return ret_val;
} /* tftp_open_cb */
```

t_tftp_close_cb

The **t_tftp_close_cb** definition specifies the format of the callback function that may be called to close a file.

Format

```
typedef void ( * t_tftp_close_cb ) ( const uint32_t hdl )
```

Arguments

Parameter	Description	Type
hdl	The file handle returned by **t_tftp_open_cb()**.	uint32_t

Return Codes

Code
None.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static void tftp_close_cb ( const uint32_t hdl )
{
    f_close( g_file[hdl] );
    g_file[hdl] = (F_FILE *)NULL;
} /* tftp_close_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static void tftp_close_cb ( const uint32_t hdl )
{
    HCC_UNUSED_ARG( hdl );
    g_buf_used = FALSE;
} /* tftp_close_cb */
```

t_tftp_abort_cb

The **t_tftp_abort_cb** definition specifies the format of the callback function that may be called to abort a connection.

After a connection is aborted, the file must be closed (and possibly removed).

The local return value identifies the reason for the abort condition. The error code and message give information on the TFTP error code sent/received.

Format

```
typedef void ( * t_tftp_abort_cb ) (
    const uint32_t          hdl,
    const char_t * const    p_filename,
    const t_tftp_mode       mode,
    const t_tftp_ret        lret,
    const t_tftp_errc       errc,
    const char_t * const    p_errmsg )
```

Arguments

Parameter	Description	Type
hdl	The directory pathname.	uint32_t
p_filename	A pointer to the name of the file.	char_t *
mode	The file open mode.	t_tftp_mode
lret	The local return value, identifying the reason for the abort condition.	t_tftp_ret
errc	The error code returned/received.	t_tftp_errc
p_errmsg	A pointer to the error message. This gives information on the TFTP error code sent/ received.	char_t *

Return Codes

Code	Description
TFTP_SUCCESS	Successful execution.
TFTP_INVALID_CB_HDL	The connection was aborted before it could be opened. This can happen if a file was GET/PUT with **tftp_client()**.
Else	See Error Codes.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static void tftp_abort_cb ( const uint32_t hdl
                          , const char_t * const p_filename
                          , const t_tftp_mode mode
                          , const t_tftp_ret lret
                          , const t_tftp_errc errc
                          , const char_t * const p_errmsg )
{
    HCC_UNUSED_ARG( lret );
    HCC_UNUSED_ARG( errc );
    HCC_UNUSED_ARG( p_errmsg );

    f_close( g_file[hdl] );
    g_file[hdl] = (F_FILE *)NULL;

    if ( mode == TFTP_MODE_WRITE )
    {
        f_delete( p_filename );
    }
} /* tftp_abort_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static void tftp_abort_cb ( const uint32_t hdl
                          , const char_t * const p_filename
                          , const t_tftp_mode mode
                          , const t_tftp_ret lret
                          , const t_tftp_errc errc
                          , const char_t * const p_errmsg )
{
    HCC_UNUSED_ARG( hdl );
    HCC_UNUSED_ARG( p_filename );
    HCC_UNUSED_ARG( mode );
    HCC_UNUSED_ARG( lret );
    HCC_UNUSED_ARG( errc );
    HCC_UNUSED_ARG( p_errmsg );

    g_buf_used = FALSE;
} /* tftp_abort_cb */
```

t_tftp_read_cb

The **t_tftp_read_cb** definition specifies the format of the callback function that may be called to read data.

Format

```
typedef t_tftp_errc ( * t_tftp_read_cb ) (
    const uint32_t      hdl,
    uint8_t * const     p_buf,
    const uint16_t      buf_len,
    uint16_t * const    p_rd_len )
```

Arguments

Parameter	Description	Type
hdl	The file handle.	uint32_t
p_buf	Where to read data from.	uint8_t *
buf_len	The length of the buffer.	uint16_t
p_rd_len	Where to write the length read.	uint16_t *

Return Codes

Code	Description
TFTP_SUCCESS	Successful execution.
Else	See Error Codes.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static t_tftp_errc tftp_read_cb ( const uint32_t hdl
                                , uint8_t * const p_buf
                                , const uint16_t buf_len
                                , uint16_t * const p_rd_len )
{
    *p_rd_len = (uint16_t)f_read( p_buf, 1, buf_len, g_file[hdl] );

    return TFTP_ERRC_SUCCESS;
} /* tftp_read_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static t_tftp_errc tftp_read_cb ( const uint32_t hdl
                                , uint8_t * const p_buf
                                , const uint16_t buf_len
                                , uint16_t * const p_rd_len )
{
    uint16_t  rd_len;

    HCC_UNUSED_ARG( hdl );

    rd_len = buf_len;
    if ( ( g_buf_size - g_buf_pos ) < rd_len )
    {
        rd_len = g_buf_size - g_buf_pos;
    }

    if ( rd_len > 0u )
    {
        memcpy( p_buf, &( g_buf[g_buf_pos] ), rd_len );
        g_buf_pos += rd_len;
    }

    *p_rd_len = rd_len;

    return TFTP_ERRC_SUCCESS;
} /* tftp_read_cb */
```

t_tftp_write_cb

The **t_tftp_write_cb** definition specifies the format of the callback function that may be called to write data.

Format

```
typedef t_tftp_errc ( * t_tftp_write_cb ) (
    const uint32_t      hdl,
    uint8_t * const     p_buf,
    const uint16_t      buf_len,
    uint16_t * const    p_wr_len )
```

Arguments

Parameter	Description	Type
hdl	The file handle.	uint32_t
p_buf	Where to write data to.	uint8_t *
buf_len	The length of the buffer.	uint16_t
p_wr_len	On return, the length of the data written.	uint16_t *

Return Codes

Code	Description
TFTP_SUCCESS	Successful execution.
Else	See Error Codes.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static t_tftp_errc tftp_write_cb ( const uint32_t hdl
                                 , uint8_t * const p_buf
                                 , const uint16_t buf_len
                                 , uint16_t * const p_wr_len )
{
    *p_wr_len = (uint16_t)f_write( p_buf, 1, buf_len, g_file[hdl] );

    return TFTP_ERRC_SUCCESS;
} /* tftp_write_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static t_tftp_errc tftp_write_cb ( const uint32_t hdl
                                 , uint8_t * const p_buf
                                 , const uint16_t buf_len
                                 , uint16_t * const p_wr_len )
{
    uint16_t  wr_len;

    HCC_UNUSED_ARG( hdl );

    wr_len = buf_len;
    if ( ( BUF_SIZE - g_buf_pos ) < wr_len )
    {
        wr_len = BUF_SIZE - g_buf_pos;
    }

    if ( wr_len > 0u )
    {
        memcpy( &( g_buf[g_buf_pos] ), p_buf, wr_len );
        g_buf_pos += wr_len;
        g_buf_size += wr_len;
    }

    *p_wr_len = wr_len;

    return TFTP_ERRC_SUCCESS;
} /* tftp_write_cb */
```

t_tftp_seek_cb

The **t_tftp_seek_cb** definition specifies the format of the callback function that may be called to search in a file opened for read.

Format

```
t_tftp_errc tftp_seek_cb (
    const uint32_t   hdl,
    const uint32_t   pos )
```

Arguments

Parameter	Description	Type
hdl	The internal handle returned by **t_tftp_open_cb()**.	uint32_t
pos	The seek position.	uint32_t

Return Codes

Code	Description
TFTP_SUCCESS	Successful execution.
Else	See Error Codes.

Examples

The code shown in these examples is given in the **ip_app_tftp_demo** package.

This example from the file **tftp_user_fs.c** shows the code interfacing to a file system:

```
static t_tftp_errc tftp_seek_cb ( const uint32_t hdl
                                , const uint32_t pos )
{
    t_tftp_errc  ret_val;

    ret_val = TFTP_ERRC_UNDEFINED;
    if ( f_seek( g_file[hdl], pos, F_SEEK_SET ) == F_NO_ERROR )
    {
        ret_val = TFTP_ERRC_SUCCESS;
    }

    return ret_val;
} /* tftp_seek_cb */
```

This example from the file **tftp_user_buf.c** shows the code interfacing to a RAM buffer:

```
static t_tftp_errc tftp_seek_cb ( const uint32_t hdl
                                , const uint32_t pos )
{
    HCC_UNUSED_ARG( hdl );
    g_buf_pos = pos;

    return TFTP_ERRC_SUCCESS;
} /* tftp_seek_cb */
```

5.5 OS Abstraction Layer

The module uses the OS Abstraction Layer (OAL) that allows it to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The module uses the following OAL components:

OAL Resource	Number Required
Tasks	1
Mutexes	1
Events	1

5.6 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.
psp_strncpy()	psp_base	psp_string	Copies one string of defined length to another.
psp_strnlen()	psp_base	psp_string	Gets the length of a string.

5.7 Utilities

The TFTP Server code creates and uses a single timer in the **hcc_timer** module.

The **hcc_timer** module is included in your system when you install the base TCP/IP modules.

6 Using the Demo Package

This module demonstrates the TFTP module. It interfaces to a FAT file system.

The test works using memory buffers (**tftp_user_buf.c**) or a file system (**tftp_user_fs.c**) as well. One source file is used at a time.

If the file system is used, the file **upl1.html** is created on the file system's root directory. This file is uploaded to the server using PUT; its content does not matter.

6.1 Demo Source Files

There are three files in the **demo** directory:

File	Description
tftp_user.h	Header file that declares the API functions.
tftp_user_buf.c	Source code of the API functions.
tftp_user_fs.c	Contains the single demo configuration option and other elements.

6.2 Demo Configuration Option

Set the single demo configuration option in the file **tftp_user_fs.c**.

MAX_FILES

The maximum number of files the module is allowed to use. The default is 2.

6.3 Demo Package API

There are just four functions in the demo.

tftp_user_init

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

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

Format

```
t_tftp_ret tftp_user_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	See Error Codes.

tftp_user_stop

Use this function to stop the TFTP user module.

Format

```
t_tftp_ret tftp_user_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	See Error Codes.

tftp_user_start

Use this function to start the demo module. This runs the tests.

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

Format

```
t_tftp_ret tftp_user_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	See Error Codes.

tftp_user_delete

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

Format

```
t_tftp_ret tftp_user_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
TFTP_SUCCESS	Successful execution.
TFTP_ERROR	See Error Codes.

6.4 Apache TFTP Server Setup

Testing the PUT and DELETE Methods

To test PUT and DELETE, install an Apache HTTP server on a (virtual) machine. Install and enable the WebDAV module (**mod_dav**).

> **Note:** The following configuration enables uploading of files to the server, so it is not secure.

Configure Apache version 2.4.7-1ubuntu4 on the machine as follows:

```
/etc/apache2/apache2.conf:
<Directory /var/www/html>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
    Dav On
    <Limit GET POST PUT DELETE HEAD OPTIONS>
        Order allow,deny
        Allow from all
    </Limit>
    <LimitExcept GET POST PUT DELETE HEAD OPTIONS>
        Order deny,allow
        Deny from all
    </LimitExcept>
</Directory>
```

Testing the POST and GET Methods

To test POST and GET, install a CGI script named **cgi_test.py** in the Apache server's html directory. The content of the file **cgi_test.py** is as follows:

```
#!/usr/bin/python
import cgi, cgitb
form = cgi.FieldStorage()
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')
print "Content-type: text/html\r\n\r\n"
print "<html><head><title>CGI Program</title></head>"
print "<body><h2>Hello %s %s</h2></body></html>" % (first_name, last_name)
```

6.5 Running the Tests

Start the tests by calling **tftp_user_start()**. The following test cases will run:

Method	File used	Description
1 PUT	upl1.html	Uploads the file **upl1.html** to the HTTP server.
2 GET	upl1.html	Downloads the file **upl1.html** and saves it to **get1.html**.