FTP Server User Guide

Version 1.80

For use with FTP Server versions 5.01 and above

Exported on 03/19/2019

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.
- 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.

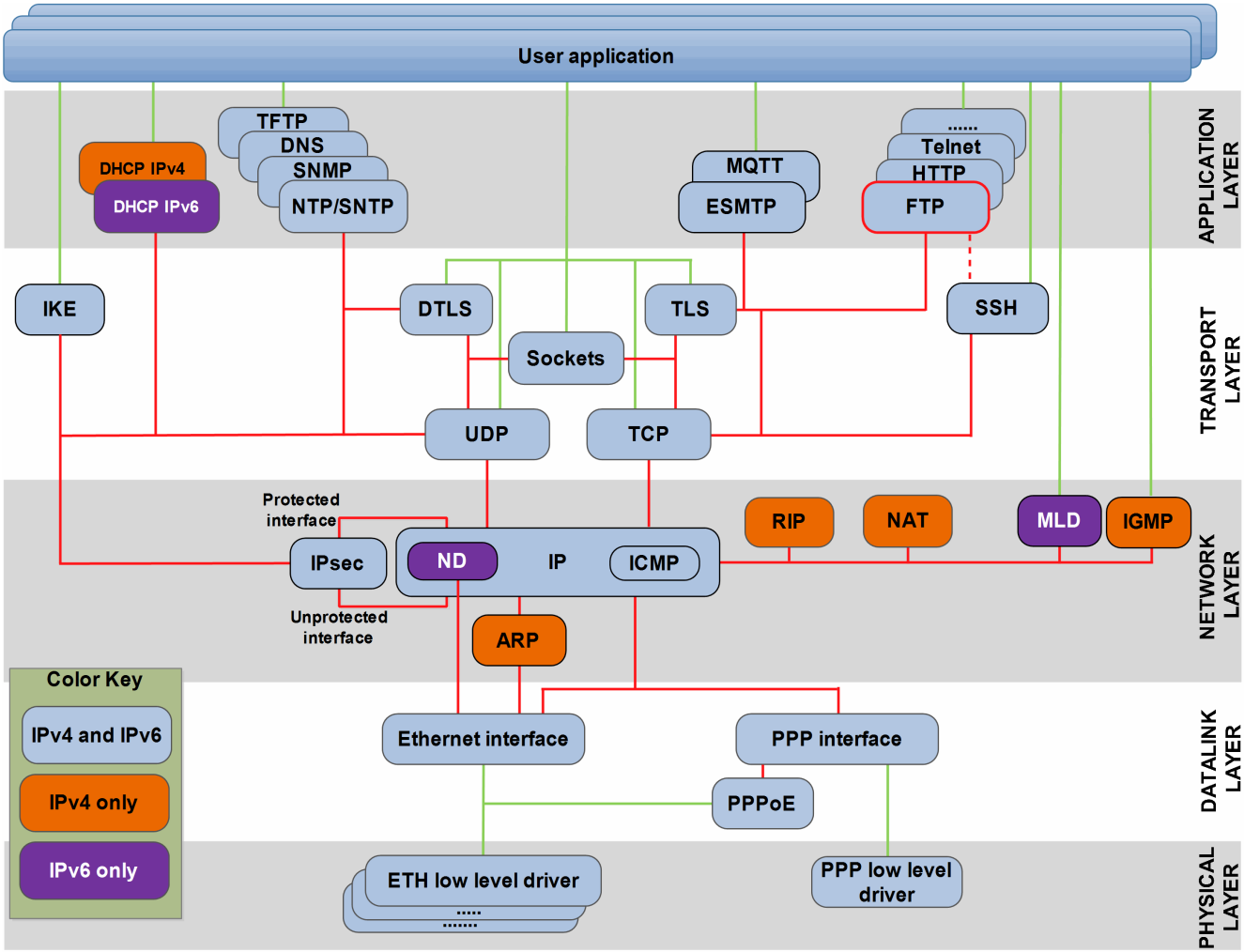
Note: To download this manual as a PDF, see TCP/IP PDFs.

1.1 Introduction

This guide is for those who want to implement an FTP Server as part of HCC Embedded's TCP/IP stack. File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet.

FTP is built on a client-server architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. HCC can also supply an FTP Client module.

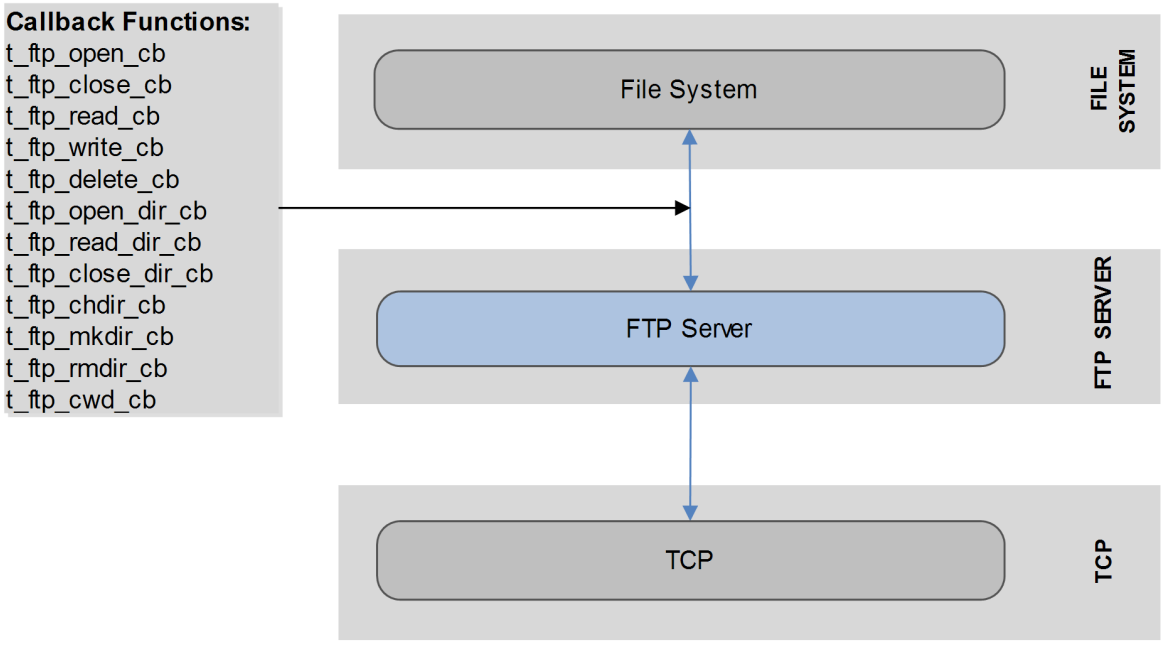
The FTP module is part of the HCC MISRA-compliant TCP/IP stack, as shown below, and is designed specifically for use with it. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



The HCC FTP 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 two types of API function:

- Start and stop the FTP module.
- Add and remove users.

In addition, a series of callback functions, listed in the following diagram, acts as the interface from the FTP server to the user's file system, as shown below. You register these dynamically using the **ftp_register_cb()** function.



You can implement as many of these callbacks as required; they must comply with the definitions described in this manual.

An FTP Server demo package is available; use this as the basis for connecting to your file system. This demo contains an implementation of the callbacks for the standard HCC file system API.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Complies with the HCC MISRA-compliant TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compliant with RFC 959.
- Abstraction layer allows connections to any file system.
- Number of simultaneous user connections is configurable.
- User authentication is configurable.
- Demo package provides sample implementations to base your FTP Server on.

1.3 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_ftp	The FTP Server package described in this manual.
ip_app_ftp_demo	The FTP Server demo package.
mip_base	The TCP/IP Dual Stack base package.
mip_tcp	The TCP 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 FTP Server User Guide

This is this document.

1.4 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_ftp.

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	2019-03-19	5.01	New template.
1.70	2017-09-04	5.01	Corrected *Packages* list.
1.60	2017-06-20	5.01	New *Change History* format.
1.50	2017-03-24	5.01	Updated network diagram. Added callback function table.
1.40	2017-01-18	4.03	Updated network diagram. Added function group tables.
1.30	2015-10-27	4.01	Extended *Introduction*.
1.20	2015-03-31	4.01	Added software *Change History*.
1.10	2014-08-19	4.01	Reorganized *System Overview* section.
1.00	2014-05-29	4.01	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 file.

2.1 API Header File

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

2.2 Configuration File

The file **src/config/config_ip_app_ftp.h** contains all the configurable FTP Server parameters. Configure these as required. For detailed explanation of these options, see Configuration Options.

2.3 FTP Server Module

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

2.4 Version

The file **src/version/ver_ip_app_ftp.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

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

FTP_SERVER_TASK_STACK_SIZE

The FTP server task stack size. The default value is 4096.

FTP_SRV_PORT

The FTP server port for command transfer. The default value is 21.

FTP_MAX_CONNS

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

FTP_MAX_DIRLEN

The maximum length of a directory character string. The default value is 64.

FTP_MAX_NUM_USER

The maximum number of users. The default value is 2.

FTP_MAX_USER_NAME_LEN

The maximum user name length. The default value is 10.

FTP_MAX_ROOT_DIR_LEN

The maximum root directory length. The default value is 10.

FTP_USE_PWD

When set, this enables password protection for users. The default value is 1.

FTP_MAX_LEN_PWD

The maximum password length. The default value is 10.

FTP_DIR_SEPARATOR

The directory separator. The default value is '/'.

FTP_CTRL_TIMEOUT

The control connection timeout in seconds. The default value is 60.

FTP_DATA_TIMEOUT

The data connection timeout in seconds. The default value is 5.

FTP_DOS

This enables saving of ASCII files in DOS format. The default value is 1.

FTP_FILENAME_MAX_LEN

The maximum length of a filename C char string. The default value is 80.

FTP_RETR_MAX_TRANSFER_LEN

The maximum transfer length a connection can send/receive at a time. The default value is 1460.

4 Application Programming Interface

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

4.1 Module Management

The functions are the following:

Function	Description
ftp_init()	Initializes the module and allocates the required resources.
ftp_start()	Starts the module.
ftp_stop()	Stops the module.
ftp_delete()	Deletes the module and releases the resources it used.

ftp_init

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

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

Format

```
t_ftp_ret ftp_init ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

ftp_start

Use this function to start the FTP Server module.

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

Format

```
t_ftp_ret ftp_start ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

ftp_stop

Use this function to stop the FTP Server module.

Format

```
t_ftp_ret ftp_stop ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

ftp_delete

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

Format

```
t_ftp_ret ftp_delete ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

4.2 Server Management

The functions are the following:

Function	Description
ftp_add_user()	Adds an FTP user.
ftp_remove_user()	Removes an FTP user.
ftp_register_cb()	Registers the set of callback functions to be used by the FTP server to access a file system.

ftp_add_user

Use this function to add an FTP user.

If user logins are not password-protected, *p_pwd* is checked and expected to be NULL.

Format

```
t_ftp_ret ftp_add_user (
    char_t *   p_name,
    char_t *   p_pwd,
    char_t *   p_root )
```

Arguments

Arguments	Description	Type
p_name	A pointer to the user name string.	char_t *
p_pwd	A pointer to the password string.	char_t *
p_root	A pointer to the root directory string belonging to the user name.	char_t *

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_INVALID_PARAM_ERR	Invalid parameter (a parameter exceeds the maximum defined in the configuration file).

ftp_remove_user

Use this function to remove an FTP user.

Format

```
t_ftp_ret ftp_remove_user ( const char_t * const p_name )
```

Arguments

Arguments	Description	Type
p_name	A pointer to the user name string.	char_t *

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
FTP_INVALID_PARAM_ERR	User not found.

ftp_register_cb

Use this function to register the set of callback functions to be used by the FTP server to access a file system.

> **Note:** A complete implementation of this function is contained in the FTP demo package that connects the server to an HCC file system. You can replace this with similar calls to your chosen file system.

Format

```
t_ftp_ret ftp_register_cb ( const t_ftp_cb_dsc * const p_cb_dsc )
```

Arguments

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

Return Values

Return value	Description
FTP_SUCCESS	Successful execution.
Else	See Error Codes.

4.3 Error Codes

If a function executes successfully, it returns with FTP_SUCCESS. The following table shows the meaning of the FTP Server return codes.

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

Return Value	Value	Description
FTP_SUCCESS	0	Successful execution.
FTP_RETRY	1	Retry.
FTP_ERR_NO_MORE_ENTRY	2	There are no more entries.
FTP_INVALID_PARAM_ERR	3	A parameter is invalid.
FTP_ERROR	4	General error.

4.4 Types and Definitions

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

Directory Entry Attributes

There are two directory entry attributes, as follows:

Name	Value	Description
FTP_ATTR_FILE	1	File.
FTP_ATTR_DIR	2	Directory.

File Open Modes

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

Name	Description
FTP_MODE_READ	Text read.
FTP_MODE_READ_BIN	Binary read.
FTP_MODE_WRITE	Text write.
FTP_MODE_WRITE_BIN	Binary write.

t_ftp_cb_dsc

The *t_ftp_cb_dsc* structure is the callback functions descriptor.

Element	Type	Description
fcd_open_cb	t_ftp_open_cb	The open callback.
fcd_close_cb	t_ftp_close_cb	The close callback.
fcd_read_cb	t_ftp_read_cb	The read callback.
fcd_write_cb	t_ftp_write_cb	The write callback.
fcd_delete_cb	t_ftp_delete_cb	The delete callback.
fcd_open_dir_cb	t_ftp_open_dir_cb	The open directory callback.
fcd_read_dir_cb	t_ftp_read_dir_cb	The read directory callback.
fcd_close_dir_cb	t_ftp_close_dir_cb	The close directory callback.
fcd_chdir_cb	t_ftp_chdir_cb	The change directory callback.
fcd_mkdir_cb	t_ftp_mkdir_cb	The create directory callback.
fcd_rmdir_cb	t_ftp_rmdir_cb	The delete directory callback.
fcd_cwd_cb	t_ftp_cwd_cb	The get current working directory callback.
fcd_rename_cb	t_ftp_rename_cb	The rename file or directory callback.

Callback Functions

These functions provide the interface between the FTP server and your file system. You can implement these as required.

Note:

- All callback functions are called from the same FTP Server task context and are protected against concurrent calls. That is, one callback function cannot interrupt another.
- There is sample code for these callbacks in the **ip_app_ftp_demo** package.

The functions are the following:

Function	Description
t_ftp_open_cb()	Specifies the format of the callback that may be called to open a file for read or write.
t_ftp_close_cb()	Specifies the format of the callback that may be called to close a file.
t_ftp_read_cb()	Specifies the format of the callback that may be called to read data.
t_ftp_write_cb()	Specifies the format of the callback that may be called to write data.
t_ftp_delete_cb()	Specifies the format of the callback that may be called to delete a file.
t_ftp_rename_cb()	Specifies the format of the callback that may be called to rename a file or directory.
t_ftp_open_dir_cb()	Specifies the format of the callback that may be called to open a directory to read entries in it that match a string.
t_ftp_read_dir_cb()	Specifies the format of the callback that may be called to read all matching entries from the directory defined by using **t_ftp_open_dir_cb()**.
t_ftp_close_dir_cb()	Specifies the format of the callback that may be called to close a directory opened with **t_ftp_open_dir_cb()**.
t_ftp_mkdir_cb()	Specifies the format of the callback that may be called to create a directory.
t_ftp_rmdir_cb()	Specifies the format of the callback that may be called to delete a directory.
t_ftp_cwd_cb()	Specifies the format of the callback that may be called to get the current working directory.

t_ftp_open_cb

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

Format

```
typedef t_ftp_ret ( * t_ftp_open_cb ) (
    char_t * const     p_filename,
    const t_ftp_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 file open mode.	t_ftp_mode
p_hdl	On return, where to write the internal handle.	uint32_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_close_cb

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

Format

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

Arguments

Parameter	Description	Type
hdl	The file handle.	uint32_t

Return Codes

None.

t_ftp_read_cb

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

Format

```
typedef t_ftp_ret ( * t_ftp_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	On return, where to write the read length.	uint16_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_write_cb

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

Format

```
typedef t_ftp_ret ( * t_ftp_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
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_delete_cb

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

Format

```
typedef t_ftp_ret ( * t_ftp_delete_cb ) ( char_t * const p_filename )
```

Arguments

Parameter	Description	Type
p_filename	The file name.	char_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_rename_cb

The **t_ftp_rename_cb** definition specifies the format of the callback function that may be called to rename a file or directory.

Format

```
typedef t_ftp_ret ( * t_ftp_open_cb ) (
    char_t * const   p_old,
    char_t *         p_new )
```

Arguments

Parameter	Description	Type
p_old	A pointer to the old filename/directory name.	char_t *
p_new	Where to write the new filename/directory name.	char_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_open_dir_cb

The **t_ftp_open_dir_cb** definition specifies the format of the callback function that may be called to open a directory to read entries in it that match a string.

The string can contain a pathname and/or wildcards. For example "*.*" searches for all files.

Format

```
typedef t_ftp_ret ( * t_ftp_open_dir_cb ) (
    char_t * const     p_name,
    uint32_t * const   p_hdl )
```

Arguments

Parameter	Description	Type
p_name	A pointer to the search string.	char_t *
p_hdl	Where to write the internal handle.	uint32_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_read_dir_cb

The **t_ftp_read_dir_cb** definition specifies the format of the callback function that may be called to read all matching entries from the directory defined by using **ftp_open_dir_cb()**.

The function continues reading until *p_buf* is full.

Format

```
typedef t_ftp_ret ( * t_ftp_read_dir_cb ) (
   const uint32_t   hdl,
   uint8_t *        p_buf,
   uint16_t         buf_size,
   uint16_t *       p_buf_len,
   uint8_t          b_only_filenames );
```

Arguments

Parameter	Description	Type
hdl	The file handle.	uint32_t
p_buf	On return, where to write the ASCII listing.	uint8_t *
buf_size	The maximum length of *p_buf* (FTP_FILENAME_MAX_LEN).	uint16_t
p_buf_len	Where to write the size of *p_buf*.	uint16_t *
b_only_filenames	One of the following: • TRUE: NLST command was received, list of filenames is needed. • FALSE: LIST command was received, Unix listing is needed.	uint8_t

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_close_dir_cb

The **t_ftp_close_dir_cb** definition specifies the format of the callback function that may be called to close a directory opened with **t_ftp_open_dir_cb()**.

Format

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

Arguments

Parameter	Description	Type
hdl	The file handle.	uint32_t

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_chdir_cb

The **t_ftp_chdir_cb** definition specifies the format of the callback function which may be called to change directory.

Format

```
typedef t_ftp_ret ( * t_ftp_chdir_cb ) ( char_t * const p_path )
```

Arguments

Parameter	Description	Type
p_path	The path to the new directory.	char_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_mkdir_cb

The **t_ftp_mkdir_cb** definition specifies the format of the callback function that may be called to create a directory.

Format

```
typedef t_ftp_ret ( * t_ftp_mkdir_cb ) ( char_t * const p_path )
```

Arguments

Parameter	Description	Type
p_path	A pointer to the directory pathname.	char_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_rmdir_cb

The **t_ftp_rmdir_cb** definition specifies the format of the callback function that may be called to delete a directory.

Format

```
typedef t_ftp_ret ( * t_ftp_rmdir_cb ) ( char_t * const p_path )
```

Arguments

Parameter	Description	Type
p_path	A pointer to the directory pathname.	char_t *

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

t_ftp_cwd_cb

The **t_ftp_cwd_cb** definition specifies the format of the callback function that may be called to get the current working directory.

Format

```
typedef t_ftp_ret ( * t_ftp_cwd_cb ) (
    char_t * const   p_path,
    const uint32_t   path_len )
```

Arguments

Parameter	Description	Type
p_path	On return, a pointer to the directory pathname.	char_t *
path_len	The maximum length of *p_path* (FTP_MAX_DIRLEN).	uint32_t

Return Codes

Code	Description
FTP_SUCCESS	Successful execution.
FTP_ERROR	Operation failed.

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 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

5.2 Utilities

The FTP 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.

5.3 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_strncat()	psp_base	psp_string	Appends a string.
psp_strncpy()	psp_base	psp_string	Copies one string of defined length to another.
psp_strncmp()	psp_base	psp_string	Compares two strings of defined length.
psp_strnlen()	psp_base	psp_string	Gets the length of a string.