InterNiche's embFTP User's Guide for PIC32MX/MZ (MPLABX Tools)

51 E. Campbell Ave
Suite 160
Campbell, CA. 95008
Copyright ©2013
InterNiche Technologies Inc.
email: Sales@iNiche.com
support: http://www.iniche.com/support

Revised: November 6, 2013

Table of Contents

Overview

This technical reference manual is provided with the InterNiche **embFTP** library. The purpose of this document is to provide enough information so that a moderately experienced "C" programmer with a reasonable understanding of TCP/IP protocols can develop FTP-based server applications using MPLABX development tools.

The primary features of this library are:

- Small footprint

- "Device Locked" to PIC32MX/MZ **Important Note:** The software described in this document will not run on any component other than the PIC32MX/MZ. For support of another controller, contact InterNiche Sales: Sales@iNiche.com
- Supports a configurable number of concurrent connections.
- Sample Applications
- Example Menu System and Command Line Interface
- DEBUG and Non-DEBUG versions of the library are provided.
- Requires use of embTCP™
- Requires existence of your own embedded file system.

A Note About this Document

Unless specifically mentioned otherwise, the term **`embTCP`** is intended to apply to both the `embTCP` and `embDUAL` embedded library products.

---

embFTP

embFTP consists of `libembftp.a`, `libembftp-debug.a`, `ftpdata.c`, `ftpdata.h`, `demo_vfs.c`, a sample application called "`example1` and a licence object called "`ftp_unregistered.o`".

embFTP's API is implemented as a set of direct and callback functions. Direct functions are implemented within embFTP and are invoked directly by the application. Access to these functions is also made available as menu commands.

Product Requirements

Installation

Before you start using this product, it is important that you have successfully built, downloaded and executed some small program using InterNiche's TCP/IP Library to your PIC32MX/MZ based board using MPLABX development tools. This is so that you have some end-to-end experience with your entire development environment and that you have confidence that your Ethernet and TCP/IP stack works.

Product Registration

As provided, embFTP contains license information that will only allow it to operate for a finite period of time before halting. Registration is accomplished by visiting www.TCPIPStack.com, submitting a simple form and checking your email for a `ftp_license.o` file that should be used instead of `ftp_unregistered.o`

Project Integration

1. Begin with a working embTCP or embDUAL project ('debug' mode)
2. Unzip the package in to the same directory that contains the embsrc, emblibs and emb_h directories

3. Add embFTP to your project:
    1. edit `tcpdata.c` file and make the following modifications:
        - Add the following line early in the file:
          `extern struct net_module ftp_module;`
        - Find the in_modules array and add the following line:
          `&ftp_module,`
        - Verify that the file "`inmain.c`" contains a call to "`example_init()`" between the call to `nichestack_init()` and `TK_START_OS()`.
    2. Add `emblibs/embFTP-debug.a` to your project
    3. Include the file `ftpdata.c` into your existing project
    4. Include the file `ftp_license.o` ** into your existing project

** NOTE: If you have not yet registered your product, use `ftp_unregistered.o`. Registration will enable full use of the library and is accomplished by visiting www.TCPIPStack.com.

This will create an operating system task for `embFTP` and will integrate it with the protocol stack.

Installing the Example1 Demo Application

`Example1` is a very simple application that uses embFTP. It is designed to show that embFTP is functioning and that a remote system can connect to it. Once this is done, it should be removed from your project.

`Example1` assumes that it is linked with a DEBUG version of `embTCP` and that the system has a console.

The file "`example1.c`" provides a fairly thorough implementation of this package, including initialization activities, callback registration and actions to be taken when a status change occurs on a connection. ftpdata.c and demo_vfs.c demonstrate the mapping of filesystem calls to your embedded file system.

To add Example1:

1. If your existing project contains references to the embTCP example, remove them from your project
2. Add ftp_examples/example1.c and demo_vfs.c to your project
3. Add embsrc/ftpdata.c to your project
4. Build and download the resulting image
5. Start your application. Once the system begins to execute, it will display on your configured output device a message similar to the following:

```
embFTP - Licensed for CHIPNAME. 0000-0000-0000-0000
Licensed to: NAME, user@example.com, COMPANYNAME
For PRODUCTNAME on CHIPNAME
```

6. From another system open an FTP client application and connect to the IP address of the running embTCP and embFTP. Example1 provides access using username "guest" and password "guest" (no quote characters, of course).

```
"220 Service ready. Welcome to InterNiche embFTP Server ";
```

Sample Application Walkthrough

Interaction between an FTP client, embFTP library, example1.c and ftpdata.c:

The embFTP task creation, global, connection, sessions and application statistics initialization will all be done at the initialization time. After the initialization, the FTP server will be listening to the FTP client connections.

`example1.c`, the FTP example application, will communicate with the FTP clients through FTP library using the embFTP API. embFTP will invoke functions in `ftpdata.c` which in turn will call functions in `example1.c`. `example1.c` maintains a database of all active clients in the form of an array of structures. This structure is called `ftpApp_connstats` and the global array variable is `ftpAppstats`. Both of these are declared in `example1.c`. The maximum number of clients is based on the constant `MAX_FTP_SESS`, defined in `ftpdata.h`.

The embFTP server maintains a similar database which is allocated at the initialization time. The maximum number of simultaneous clients for is specified by the `max_ftp_conn` variable which initialized to `MAX_FTP_SESS`. This variable itself is declared in `ftpdata.c`. To change the number of supported connections, simply change `MAX_FTP_SESS` and recompile your application.

Provided that fewer than `max_ftp_conn` active connections exist, whenever a client establishes an FTP connection with embFTP, it will allocate memory for the connection on the heap and assign a connection id. It will then transmit the login prompt to the client.

Once the client provides the username and password, the embFTP server will invoke the `validate_ftpUser()` function found in `ftpdata.c`. If appropriate, this function can check if the client ip address is authorized by calling function `check_ftpipAddr()` in `example1.c`. The function will then validate the username and password provided by the client. If found to be invalid, then the client will be allowed `ftp_max_login_tries` (`ftpdata.c`) login attempts before embFTP terminates the connection and the client will have to reconnect. Once the valid username and valid password have been provided, callback functions will be registered (`ftpApp_callbcks_reg()`). This function will register two callback functions `ftpApp_connsts()` and `ftpApp_errRecv()`, both of these functions can be found in `example1.c`.

If `validate_ftpUser()` function returns `success`, embFTP will invoke the `ftpApp_connsts()` function in `example1.c` with status set to `FTP_SESS_OPENED`. If the call to `ftpApp_connsts()` succeeds, embFTP will present '`230 User logged in`' to the ftp client.

When this occurs, `example1.c` will assign an entry to the connection in its `ftpAppstats` table and will store `conn_state`, `conn_ID` and `username` and embFTP will be ready to accept commands from the client.

embFTP will close a client session under the following circumstances:

1. Normal termination by the FTP client
2. `ftp_idle_time_enable` was set and the client was idle for `ftp_conntmo` seconds (see `ftpdata.c`).
3. The embFTP application closed the connection through a call to `ftp_closeconn()` either through a menu command or API call.

Whenever a connection is closed, embFTP will invoke the `ftpApp_connsts()` function with status set to `FTP_SESS_CLOSED`. When this occurs in `example1.c`, the function clears the appropriate entries in the `ftpAppstats` table.

Based on the setting of `MSGS_TO_CONSOLE` (`ftpdata.h`), embFTP will display messages on the console or the serial port for the following actions:

1. successful login;
2. call_back registration;
3. user logout.

Using the Example

`example1.c` is supported by a RAM-based demo file system supporting only four pseudo-files:

1. `ftp_rdonly_bin.bin`: a 2 Megabyte READ-ONLY binary file.
2. `ftp_wronly_bin.bin`: a WRITE-ONLY binary file with a maximum size of 2 Megabytes.
3. `ftp_rdonly_ascii.txt`: a 200 byte READ-ONLY ascii file.
4. `ftp_wronly_ascii.txt`: a WRITE-ONLY ascii file with a maximum size of 200 bytes.

These files can be manipulated and examined by appropriate use of an FTP Client's `GET`, `PUT` and `DIR` commands.

Debug vs Non-Debug Libraries

embFTP includes two versions of the library: Debug, which is intended for use during initial application development; and Non-Debug, which is appropriate for use in your final product.

An Important Note Regarding Stack Sizes

It is important to recognize that the task stack size requirements must be set appropriately for the unique requirements of your application and the requirements of your final product. Failure to properly tune the stacks will result in either wasted memory or nearly impossible to diagnose runtime errors.

The size of embFTP's task stacks are specified in the `ftpdata.c` file. Please refer to FreeRTOS.org for information regarding stack sizing and the debugging of stack overflow conditions.

Configuration

Tunable and Reference Parameters

The `ftpdata.c` file contains a set of parameters that may be tuned for the specific implementation. These are shown in the following table and their values are set in `ftpdata.h`.

Type	Parameter	Default Value	Description
uint16_t	ftp_stksize	3072	Default size in bytes of embFTP's OS stack
int	max_ftp_conn	MAX_FTP_SESS	Default max no. of ftp sessions
int	ftp_port	FTP_PORT	Default ftp port
int	ftp_max_login_tries	FTP_MAX_LOGIN_TRIES	Default ftp max login tries
int	ftp_conntmo	FTP_IDLE_TIME	Default ftp idle time

int	ftp_idle_time_enable	FTP_IDLE_TMO	ftp idle timeout enable
char	ftp_slash	FTP_SLASH	ftp path delimiter
char *	default_drive	DEFAULT_DRIVE	default drive
int	drive_letters	DRIVE_LETTERS	drive letters enable
int	filebuf_size	FILEBUFSIZE	default file send and recv buf size
char *	ftpclient_banner	"220 Service ready. Welcome to InterNiche embFTP Server "	String presented to client upon successful connection establishment

From ftpdata.h:

```
#define  MAX_FTP_SESS            2     /* Default max no. of ftp sessions */
#define  FTP_PORT                21    /* ftp port    */
#define  FTP_MAX_LOGIN_TRIES     5     /* ftp client max no. of login tries */
#define  FTP_IDLE_TIME           120   /* ftp idle time after which connection is deleted */
#define  FTP_IDLE_TMO            1     /* enabling the session timeout */
#define  FILEBUFSIZE             1024  /* ftp socket send/recv file buffer size  */

/* set up file system options for target system */

#define  FTP_SLASH               '\\'  /* use UNIX style slash('/' and '\\' for DOS file
systems OR leave it empty */
#define  DEFAULT_DRIVE           " "   /* Default base directory, can be "c:" OR  empty " "
*/
#define  DRIVE_LETTERS           0     /* '0' indicates disabled, which sets default_drive to
" " */
```

API

The embFTP API is implemented as a set of functions falling into two categories:

1. Direct functions are implemented at the FTP server. These functions can be invoked by the application. Direct function prototypes are available in ftpdata.h. These functions are also implemented in the form of menu commands.
2. Callback functions are to be implemented by the application developer. For examples, refer to `example1.c`.

Direct functions:

Name

`ftp_startup()`

Syntax

```
int ftp_startup(void );
```

Parameters

none

Description

At the initialization of the ftp module, this function is called. When this function is called, ftp server task would have been created and would be in suspended state.

This function will initialize all the connections and statistics at the server and start listening to ftp client connections. Then it will resume the ftp server task. This function sets a flag to indicate that this function is called and ftp server is resumed.

This is an embFTP library function. This function is implemented as a menu option(ftpstartup) at the console. This menu option is associated with embftp group.

Returns

SUCCESS (`0`) or an error code

Example

```
{
   int err  = 0;
   err = ftp_startup();
   if (err  != 0)
      printf("App:Error in ftp_startup() \n");
}
```

Notes

- Refer to the Man pages for the syntax to execute this function as menu command .

Name

`ftp_shutdown()`

Syntax

```
void ftp_shutdown(void);
```

Parameters

none

Description

It will delete all the active connections and set shut_down_pending flag and signal the ftp task. Task suspension will be done inside the task body after the task checks that the flag is set and there are no active connections.

This function can be invoked by the application.

This is an embFTP library function. This function is implemented as a menu option(ftpshutdown) at the console . This menu option is associated with embftp group.

Returns

nothing

Example

```
{
   ftp_shutdown();
}
```

Notes

- Refer to the Man pages for the syntax to execute this function as menu command .

Name

`ftp_closeconn()`

Syntax

```
int ftp_closeconn(int ftpconnID);
```

Parameters

ftpconnID	This is the connection id provided in the `ftpApp_connsts()` callback function

Description

This function frees memory that was allocated for the FTP connection. It deletes the object from the queue of open ftp connections and also closes the underlying TCP connection. It also closes any file if it is open still. It may be called by the application.

This is an embFTP library function. This function is available as a menu option (ftpdelconn) at the console. This menu option is associated with ftp group.

With the menu option, either a single connection can be deleted or all the active connections can be deleted.

Returns

SUCCESS (`0`) or an error code

Example

```
{
   int err  = 0;
   int conn_id = 2;

   err = ftp_closeconn(conn_id);
   if (err  != 0)
      printf("App:Error in ftp_closeconn() \n");
}
```

Notes

- Refer to the Man pages for the syntax to execute this function as menu command .

Name

`ftp_get_stats()`

Syntax

```
int ftp_getstats(struct Ftp_API_Stats *statsArea);
```

Parameters

statsArea	Memory address at which the statistics will be written

Description

This function can be called by the application. This function copies global and session statistics to the memory provided by the calling application. Common stats will have number of opened connections, number of closed connections, number of active connections. user session will have the information total bytes received, total no of commands executed, total number of bytes sent, if the session is open/closed, the session duration in ticks, Session connectionID, ip address and the username.

This is an embFTP library function. This is accessible as a menu option (ftpstats). This menu option is associated with ftp group.

Returns

SUCCESS (`0`) or an error code

Example

```
{
   struct Ftp_API_Stats *stats;

   stats = (struct Ftp_API_Stats *)malloc(sizeof (struct Ftp_API_Stats)); /* Assuming malloc
() memory alloc function used */
   if (stats)
   {
      err  = ftp_get_stats(stats);
      printf( "Total connections opened = %ld\n", stats->ftpStats.conn_opened);
   }
   free (stats);  /* Assuming free() is memory dealloc function used */
}
```

Notes

- The calling application will have to allocate the memory and pass the pointer to this function.
- Structure Ftp_API_Stats and associated structures are in the file, ftpdata.h.

Name

`ftp_callbckfn_reg()`

Syntax

```
int ftp_callbckfn_reg(int connID, int code_type, int (*func_ptr)(int, void *));
```

Parameters

connectionID	This is the connection id provided in the `ftpApp_connsts()` callback function
code_type	either CONN_STS or ERR_CODE
func_ptr	address of function being registered with embFTP

Description

This is an embFTP library function. This function registers a call back function. Currently there is option to register two call back functions per ftp session. Registration of both of them is shown in example1.c

Returns

SUCCESS (`0`) or an error code

Example

```
{
   int err = 0;
   err = ftp_callbckfn_reg(connID, CONN_STS, &ftpApp_connsts); /* Conn status call back
registration */
}
```

Callback functions in example1.c :

These functions will be called by the ftp server.

Name

`ftpApp_connsts()`

Syntax

```
int ftpApp_connsts(int conn_ID, void *parm1);
```

Parameters

connectionID	This is the connection id provided in the `ftpApp_connsts()` callback function
param	parameter passed for this function , which has state and username

Description

FTP server invokes this function when it recognizes one of the following events occurring on a ftp client:

1. Successful login (FTP_SESS_OPENED)
2. Successful Logout (FTP_SESS_CLOSED)

Returns

SUCCESS (`0`) or an error code

Name

`ftpApp_errRecv()`

Syntax

```
int ftpApp_errRecv(int conn_ID, void *parm);;
```

Parameters

connectionID	This is the connection id provided in the `ftpApp_connsts()` callback function
param	parameter passed for this function , which has state and username

Description

The FTP server invokes this function when there is an error. This function displays the error string. An example implementation of this function is shown in example1.c

Returns

SUCCESS (`0`) or an error code

Other Functions in example1.c

Name

`example_init()`

Syntax

```
int example_init(void);
```

Parameters

none

Description

This function is called by the in_main.c module at initialization time. Put all initialization, needed by the application here. Currently, for the example application there are four files created in this function, a write only ascii file, write only binary file, a rea donly ascii file and a read only binary file.

Returns

SUCCESS (`0`) or an error code

Name

`ftpApp_init()`

Syntax

```
int ftpApp_init(void);
```

Parameters

none

Description

This function is called by the ftp server at initialization time.

Returns

SUCCESS (`0`) or an error code

Name

`check_ftpipaddr()`

Syntax

```
int check_ftpipAddr(struct sockaddr sin);
```

Parameters

sin	This is the socket address of the FTP Client

Description

If IP filtering needs to be implemented for the IP address, from where ftp clients can login to the ftp server, that can be done in this function. This function is invoked by validate_ftpUser() function in ftpdata.c

Returns

SUCCESS (`0`) or an error code

Name

`ftpApp_callbcks_reg()`

Syntax

```
int ftpApp_callbcks_reg(int connID,char *username);
```

Parameters

connID	This is the connection id provided in the `ftpApp_connsts()` callback function
username	This is the user name assiciated with this FTP session.

Description

This function registers the Call Back functions for the ftp application. There are two call back functions, one for the connection status and the other for error code. This function is invoked by validate_ftpUser() function in ftpdata.c

Returns

SUCCESS (`0`) or an error code

Functions in ftpdata.c

All functions in `ftpdata.c` should be modified according to your specific requirements. These functions fall into two categories: user validation and file system mapping.

As provided, the file system mapping functions are supported by a simple pseudo-file system implemented by `ftp_example/demo_vfs.c`.

User Validation Function

Name

`validate_ftpUser()`

Syntax

```
int validate_ftpUser(int connID,char *username, char *password, struct sockaddr sin);
```

Parameters

connectionID	This is the connection id

username	ftp username to be validated
password	ftp password to be validated
sin	client socket address

Description

This is called from the ftp server. When a user logs in at ftp client, ftp server invokes this function. FTP Application is expected to validate the username and password. Here a very simple user validation is implemented verifying if the username and password both are guest. If the user validation is success, this function does the following:

- Invoke callbacks registration function. /* In example1.c */
- Invoke a function which can implement ip filtering for the ftp client ip address /* In example1.c */

Returns

SUCCESS (`0`) or an error code

File System Mapping functions

`embFTP` interacts with your file system through 10 function calls, which must be modified to make use of your specific file system.

- int vf_init(void);
- void *vfopen(char *filename, char *mode);
- void vfclose(void *fptr);
- int vfread(char *buf, unsigned size, unsigned count, void *fptr);
- int vfwrite(char *buf, unsigned size, unsigned items, void *fptr);
- int vfgetc(void *fptr);
- int vunlink(char * name);
- int fs_curwrkdir(char *drive, char *cwd, char *buf); /* current working directory */
- int fs_dodir(void *ftp, char *dirname, char *buf);
- int fs_chwrkdir(char *drive, char *cwd, char *path, char *filename, void *ftp, char *buf);

Name

`vf_init()`

Syntax

```
int vf_init(void);
```

Parameters

none

Description

Does any file system initialization that might be required. Add any needed file system initialization here

Returns

SUCCESS (`0`) or an error code

Name

`vfopen()`

Syntax

```
void *vfopen(char *filename, char* mode);
```

Parameters

filename	This is the filename of the file to be opened.
mode	mode in which the file needs to be opened.

Description

It opens a file whose name is in the parameter filename and the mode is in mode parameter. Mode can be r-- ASCII read, rb-- binary read, w-- ASCII write and wb-- binary write.

Returns

Pointer to open void structure

FAT/NTFS implementation example:

```
void *
vfopen(char *filename, char *mode)
{
   void *fptr;
   HT_VF_LOCK(); /* Resource Locking */
   /* Replace this with call to implementation function for your platform */
   fptr = (void *)fopen(filename, mode);
   HT_VF_UNLOCK(); /* Resource unlocking */
   return (fptr);
}
```

Name

`vfclose()`

Syntax

```
void vfclose(void *fptr);
```

Parameters

fptr	Pointer to open void structure.

Description

The fclose() function causes the stream pointed to by fptr to be flushed and the associated file to be closed.

Returns

nothing

FAT/NTFS implementation example:

```
void
vfclose(void *fptr)
{
   HT_VF_LOCK();
   /* Replace this with call to implementation function for your platform */
   fclose(fptr);
   HT_VF_UNLOCK();
}
```

Name

`vfread()`

Syntax

```
int vfread(char *buf, unsigned size, unsigned count, void *fptr);
```

Parameters

buf	buffer where data will be stored.
size	size in bytes for each data item.
count	number of data items to be read.
fptr	pointer to an open void structure.

Description

The fread() function shall read into the array pointed to by buf up to count elements whose size is specified by size in bytes, from the stream pointed to by fptr.

Returns

Number of items read.

FAT/NTFS implementation example:

```
int
vfread(char *buf, unsigned size, unsigned count, void *fptr)
{
   int rc;

   HT_VF_LOCK();

   /* Replace this with call to implementation function for your platform */
   rc = fread(buf, size, count, fptr);

   HT_VF_UNLOCK();

   return (rc);
}
```

Name

`vfwrite()`

Syntax

```
int vfwrite(char *buf, unsigned size, unsigned items, void *fptr);
```

Parameters

buf	buffer containing data to be written.
size	size in bytes for each data item.
count	number of data items to be written.
fptr	pointer to an open void structure.

Description

The fwrite() function shall write, from the array pointed to by buf, up to items elements whose size is specified by size, to the stream pointed to by fptr.

Returns

Number of items written.

FAT/NTFS implementation example:

```
int
vfwrite(char *buf, unsigned size, unsigned items, void *fptr)
{
   int rc;

   HT_VF_LOCK();

   /* Replace this with call to implementation function for your platform */
   rc = fwrite(buf, size, items, fptr);

   HT_VF_UNLOCK();
```

```
   return (rc);
}
```

Name

`vfgetc()`

Syntax

```
int *vfgetc(void *fptr);
```

Parameters

fptr	pointer to an open void structure.

Description

function will get the next character as an int, from the input stream pointed to by fptr, or EOF on failure.

Returns

character from current position of file pointer or EOF on failure.

FAT/NTFS implementation example:

```
int
vfgetc(void *fptr)
{
   int  rc;

   HT_VF_LOCK();

   /* Replace this with call to implementation function for your platform */
   rc = fgetc(fptr);

   HT_VF_UNLOCK();

   return (rc);
}
```

Name

`vunlink()`

Syntax

```
int vunlink(char *name);
```

Parameters

name	filename of the file to be deleted.

Description

deletes a file from the file system.

Returns

SUCCESS (`0`) or an error code

FAT/NTFS implementation example:

```
int
vunlink(char * name)
{
   int   rc =  0;
   HT_VF_LOCK();
   rc = remove(name);
   HT_VF_UNLOCK();
   return (rc);
}
```

Name

`fs_curwrkdir()`

Syntax

```
int fs_curwrkdir(char *drive, char *cwd, char *buf);
```

Parameters

drive	current working drive
cwd	current working directory.

Description

current working directiory command (when pwd and CR is entered at the client window.) The user of this function is expected to copy the present working drive and directory into the argument 'buf' which is passed on to it. This string will be displayed at the ftp client window, by the caller of this function.

Returns

SUCCESS (`0`) or an error code

FAT/NTFS implementation example:

```
int
fs_curwrkdir(char *drive, char *cwd, char *buf)
{
   int ret = 0;
```

```
    ret = sprintf(buf, "257 \"%s%s\"\r\n", drive, cwd);
    return (ret);
}
```

Name

`fs_chwrkdir()`

Syntax

```
int fs_chwrkdir(char *drive, char *cwd, char *path, char *dirname, void *ptf, char *buf);
```

Parameters

drive	current working drive like c:, d: etc
cwd	current working directory, without any drive spec. At the end of this function it will have the changed directory name without the drive letter.
path	the directory name to be changed to.
dirname	current working directory along with the drive letter.
ptf	Variable used internally by the embftp library - **Do Not Modify.**
buf	Buffer for the error message to be copied into, this message will be displayed onto the clients window.

Description

Change the current working directory to the directory specified in the request from the FTP client. If the requested path does not exist, the implementation of this function should pass an error message in 'buf' and return an error code.

Returns

SUCCESS (`0`) or an error code

FAT/NTFS implementation example:

If the values of the variables before the successful execution of function are as follows:

drive	c:
cwd	"\"
path	"temp"
dirname	""
ptf	0x00c19c18
buf	""

The values of the variables after the successful execution of the function should be as follows, so that the calling function in embFTP can display it correctly:

drive	c:
cwd	"\temp"
path	"temp"
dirname	"c:\temp"
ptf	0x00c19c18
buf	""

If the directory (`temp`) is NOT valid, the `buf` should contain an error message, as shown below:

drive	c:
cwd	"\"
path	"temp"
dirname	"c:\temp"
ptf	0x00c19c18
buf	0x.... "550 Unable to find c:\temp"

Name

`fs_dodir()`

Syntax

```
int fs_dodir(void *ptf, char *dirname, char *buf);
```

Parameters

ptf	Variable used internally by the embftp library. The value of `ptf` should not be modified.
dirname	Current working directory along with the drive letter.
buf	Buffer for the message to be copied into, this message will be displayed onto the clients window.

Description

fs_dodir() - when "dir" (or "ls" for UNIX) is typed at the current ftp client, this function will write the resulting text out on datasock, which is open before this is called.

The function implementation should:

1. Invoke function `dir_listing_start(ptf)` to open the data connection, if not opened already.
2. Invoke `xmit_dir_ent_txt(ptf)` once for each file in the directory, providing a formatted line of text in `buf`.
3. Invoke function `dir_listing_done(ptf)` to close the data connection.

Returns

SUCCESS (`0`) or an error code

---

embFTP Menu CLI

ftpstats - Display FTP statistics

---

ftpstats

`ftpstats - Display FTP statistics`

Syntax

`ftpstats [-c] [-s]`

Parameters

-c	Display Connection stats.
-s	Display session stats

none

Description

This command displays ftp stats. If '-c' is specified, the connection stats are displayed. If '-s' is specified, session stats are displayed. Specifying no parameters is an error.

Notes

- If no parameters are specified, it is considered an error condition.

Location

This command is provided by the Ftp module when FTPSTATS is defined in ftpdata.c

Example

```
-> ftpstats -c -s
```

ftphexdump - Display the hexdump of the embFTP statistics structure

Name

`ftphexdump - Display the hexdump of the embFTP statistics structure`

Syntax

`ftphexdump`

Parameters

None

Description

This command displays the hexdump of the embFTP statistics structure.

Location

This command is provided by the FTP module when FTPHEXDUMP is defined in `ftpdata.c`.

Example

```
-> ftphexdump
```

ftpdelconn - Delete ftp connection/connections

Name

`ftpdelconn - Delete ftp connection/connections`

Syntax

`ftpdelconn -c -n -a`

Parameters

-c	Delete a connection
-n	Connection id
-a	Delete all the active connections

Description

This command deletes a connection or all active connections.

Notes

- This command is not valid within an ftp session. It can be executed at the console only.
- Connection ids for all connections can be presented by the "`ftpstats -s`" command.

Location

This command is provided by the Ftp module when FTPDELCONN is defined in ftpdata.c.

Examples

1. To delete a specific connection:

```
-> ftpdelconn -c -n 2
```

   Where in this example "2" is the connection id.
2. To delete all the active connections:

```
-> ftpdelconn -a
```

ftpshutdown - suspend the FTP task

Name

`ftpshutdown - suspend the FTP task`

Syntax

`ftpshutdown`

Parameters

None

Description

This command suspends the FTP task.

Location

This command is provided by the Ftp module when FTPSRVSHUTDOWN is defined in `ftpdata.c`.

Example

```
-> ftpshutdown
```

ftpstartup - start the FTP task

Name

```
ftpstartup - start the FTP task
```

Syntax

```
ftpstartup
```

Parameters

None

Description

This command starts the FTP task.

Location

This command is provided by the embFTP module when FTPSRVSTARTUP is defined in ftpdata.c.

Example

```
-> ftpstartup
```

Related Products

This product was derived from a portable, flexible and more full-featured product available from InterNiche Technologies, Inc. For more information about this **SOURCE CODE PRODUCT**, please visit www.iNiche.com or email Sales@iNiche.com.

For Additional Information ...

- InterNiche Support Site
- FreeRTOS web site