FTP Client Engine

Reference Manual

(FCE_REF)

Version 4.0

October 10, 2023

This software is provided as-is.
There are no warranties, expressed or implied.

MarshallSoft Computing, Inc.

Huntsville AL 35815 USA

Voice: 1.256.881.4630
Web: http://www.marshallsoft.com

TABLE OF CONTENTS

1 Introduction

The **FTP Client Engine (FCE)** is a component library that uses the Windows API to provide direct and simple control of the FTP protocol. The FCE component library can be used for both anonymous and private FTP sessions.

A straightforward interface provides the capability to quickly develop FTP software applications to connect to any FTP server, navigate its directory structure, list files, upload files, delete files, append files, and download files using the FTP protocol.

The FCE Reference Manual (FCE_REF) contains details on each individual FCE function.

Fully functional versions of our **FTP Client** software components are provided so that the developer can test the **FCE** library in their environment. The evaluation version as well as a list of the many FTP Client library features provided can be found on our website at:

`http://www.marshallsoft.com/ftp-client-library.htm`

1.1 General Remarks

All functions return an integer code. Negative values are always errors. See Section 3 "FCE Error Return Code List". Non-negative return codes are never errors.

Note that the **fceErrorText** function is used to get the text message associated with any error code.

Each function argument is marked as:

- `(I) : 4-byte integer (Win32).`
- `(L) : 4-byte integer (Win32).`
- `(P) : 4-byte pointer (Win32).`

Refer to the declaration files (see section 1.3 below) for the exact syntax of each FCE function. Also note that the example programs show exactly how FCE functions are called.

1.2 Documentation Set

The complete set of documentation consists of three manuals. This is the third manual (FCE_REF) in the set.

- FCE4x Programmer's Manual `(FCE_4x.PDF)`
- FCE User's Manual `(FCE_USR.PDF)`
- FCE Reference Manual `(FCE_REF.PDF)`

The FCE4x Programmer's Manual is the computer language specific manual. All language dependent programming issues including installation, compiling and example programs are discussed in this manual. Language specific manuals are as follows:

FCE_4C.PDF	FCE Programmer's Manual for C/C++
FCE_4D.PDF	FCE Programmer's Manual for Delphi
FCE_4VB.PDF	FCE Programmer's Manual for Visual Basic
FCE_4PB.PDF	FCE Programmer's Manual for PowerBASIC
FCE_4FP.PDF	FCE Programmer's Manual for Visual FoxPro
FCE_4DB.PDF	FCE Programmer's Manual for Visual dBase
FCE_4XB.PDF	FCE Programmer's Manual for Xbase++

The FCE User's Manual (FCE_USR.PDF) discusses FTP in general as well as language independent programming issues such as technical support, purchasing and license information. Read this manual after reading the FCE_4x Programmer's Manual.

The FCE Reference Manual (FCE_REF.PDF) contains details on each individual FCE function.

All documentation can also be accessed online at
http://www.marshallsoft.com/support.htm.

1.3 Declaration Files

The exact syntax for calling FCE functions are specific to the host language (C/C++, Delphi, VB, etc.) and are defined for each language in the "FCE declaration files". Each FCE product comes with the appropriate declaration file for the supported language. For example,

FCE4C	C/C++ and .NET	FCE.H
FCE4VB	Visual Basic	FCE32.BAS/ FCE64.BAS
	VBA (EXCEL, ACCESS, etc.)	FCE32.BAS
FCE4PB	PowerBASIC	FCE32.PBI
FCE4D	Borland/Embarcadero Delphi	FCE32.PAS/FCE64.PAS
FCE4FP	Visual FoxPro	FCE32.FOX
FCE4DB	Visual dBase	FCE32.CC
FCE4XB	Xbase++	FCE32.CH

All FCE functions are used in one or more example programs.

1.4 Language Notes

All language versions of FCE include the example program FCEVER. Refer to this program and the declaration file as defined in Section 1.3 above to see how FCE functions are called. The FCEVER program is also the first program that should be compiled and run.

1.4.1 C/C++

None.

1.4.2 Delphi

Functions defined in the Delphi Unit FCEW.PAS begin with "f" rather than "fce".

1.4.3 Visual Basic

None.

1.4.4 PowerBASIC

Constants defined for PowerBASIC (FCE32.PBI) begin with the character '%' symbol. The FCE keycode is defined in KEYCODE.PBI.

1.4.5 Visual FoxPro

All strings passed to FCE functions must be prefixed with the '@' character.

1.4.6 Visual dBase

None.

1.4.7 Xbase++

Functions defined for Xbase++ begin with 'X'. All strings passed to FCE functions must be prefixed with the '@' character.

2 FCE Functions

fceDebug is not listed below since it is used only for internal diagnostics.

2.1 fceAbort Abort fceDriver.

SYNTAX

```
fceAbort(Channel)

       Channel : (I) Channel number
```

REMARKS

The **fceAbort** function is used to abort the FCE state driver. This is used when calling the FCE state driver (**fceDriver**) directly and it is necessary to abort.

After calling **fceAbort**, subsequent calls to **fceDriver** will return 0 (IDLE). Thus, FCE is ready for the next command.

This function is not required unless the state driver **fceDriver** is being called directly.

RETURNS

`Return < 0 : An` error has occurred. Call `fceErrorText`.

EXAMPLES

`See the WINFTP example program.`

C/C++ Example

```
    // abort FCE
    fceAbort(0);
```

BASIC Example

```
    ' abort FCE
    Code = fceAbort(0)
```

ALSO SEE

`fceDriver`

2.2 **fceAttach** Initializes FTP Client Engine

SYNTAX

```
fceAttach(NbrChans, KeyCode)

       NbrChans : (I) Number of channels or threads.
       KeyCode  : (L) Registration key code.
```

REMARKS

The **fceAttach** function must be the first FCE call made. Pass the maximum number of channels or threads that will be in use. Use NbrChans = 1 for non-threaded applications.

The 'Chan' parameter for subsequent calls to FCE functions must be in the range of 0 to NbrChans.

Up to 32 threads (numbered from 0 to 31) can be started, each of which can be connected to a different FTP server and run independently.

When FCE is registered, you will receive a 'KeyCode' that matches the 'KeyCode' within the registered DLL. For the evaluation version, the keycode is 0. See file KEYCODE.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

```
All example programs call fceAttach.
```

C/C++ Example

```
    // Initialize FCE (look in KEYCODE.H for FCE_KEY_CODE)
    fceAttach(1, FCE_KEY_CODE);
```

BASIC Example

```
    ' Initialize FCE (look in KEYCODE.BAS for FCE_KEY_CODE)
    Code = fceAttach(1, FCE_KEY_CODE)
```

ALSO SEE

```
fceRelease.
```

2.3 <u>**fceByteToShort**</u> Converts 8-bit character buffer to 16-bit

SYNTAX

```
fceByteToShort(Buffer)

       Buffer : (P) character buffer
```

REMARKS

The **fceByteToShort** function converts the (null terminated) character buffer 'Buffer' from 8-bit ASCII characters to 16-bit Unicode ASCII characters.

The buffer <u>must</u> be null terminated (last character is a hex 00) and the buffer <u>must</u> be at least twice the size (in bytes) of the character string (since 16-bit characters require twice the space as 8-bit characters).

This function is only necessary when working with 16-bit Unicode ASCII characters in C# and Delphi 2005.

RETURNS

```
None.
```

EXAMPLES

See C# example cs_get.csproj

C# Example

```
     char[] UnsafeBuffer = new char[128];
     // get the registration string
     fixed (char* pBuffer = UnsafeBuffer)
     Code = fceGetString(0, FCE_GET_REGISTRATION, pBuffer, 127);
     if(Code>0)
      {// convert (null terminated) UnsafeBuffer[] to 16-bit chars (unicode)
       fixed (char* pBuffer = UnsafeBuffer)
       fceByteToShort(pBuffer);
      }
```

ALSO SEE

```
fceShortToByte
```

2.4 **fceClose** Closes connection opened by fceConnect.

SYNTAX

```
fceClose(Channel)

       Channel : (I) Channel number
```

REMARKS

The **fceClose** function closes the connection to the FTP server opened with **fceConnect**. After closing, another connection on channel 'Chan' may be opened with **fceConnect**.

If **fceConnect** fails, do NOT call **fceClose**.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

All example programs that call **`fceConnect`** will also call **`fceClose`**.

`C/C++ Example`

```
    // close connection.
    fceClose(0);
```

`BASIC Example`

```
    ' close connection.
    Code = fceClose(0)
```

ALSO SEE

```
fceConnect.
```

2.5 <u>**fceCommand**</u> Sends arbitrary command to server.

SYNTAX

```
fceCommand(Channel, Text)

     Channel : (I) Channel number.
     Text    : (P) Command text.
```

REMARKS

The **fceCommand** function is used to send an FTP protocol command (up to 128 bytes) to the FTP server. The FTP server must recognize the command text. A non-negative return code indicates that the server has accepted the command.

Some of the FTP protocol commands that may be useful are:

`RNFR`	Rename file "from" (on server).
`RNTO`	Rename file "to" (on server)
`SYST`	Request the host operating system.
`STAT`	Request status of current file transfer.
`HELP`	Request help on supported FTP commands.
`NOOP`	No operation.

RFC 959 contains the full list of FTP protocol commands.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

C/C++ Example

```
    // rename file "oldname.txt" to "newname.txt".
    fceCommand(0, "RNFR oldname.txt");
    fceCommand(0, "RNTO newname.txt");
```

BASIC Example

```
    ' rename file "oldname.txt" to "newname.txt".
    fceCommand(0, "RNFR oldname.txt")
    fceCommand(0, "RNTO newname.txt")
```

2.6 <u>**fceConnect**</u> Connects to an FTP server.

SYNTAX

```
fceConnect(Channel, Server, User, Pass)

  Channel : (I) Channel number.
  Server  : (P) Server name or dotted IP address.
  User    : (P) Users account name or "anonymous".
  Pass    : (P) Password for above.
```

REMARKS

The **fceConnect** function connects to the FTP server 'Server' and logs on as 'User' with password 'Pass'.

FTP servers that allow anonymous access will accept "ftp" or "anonymous" for the user name and your email address for the password.

Pass a null string (a string in which the first byte is zero) for 'User', and 'User' and 'Pass' will **not** be sent to the server when connecting. Pass a null string for 'Pass', and the 'Pass' is not sent to the server. In this case, the **fceCommand** function must be used to pass any required information to the server. This is typically necessary when connecting through a proxy server.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

Most example programs call `fceConnect`.

`C/C++ Example`

```
    // Connect to FTP server
    Code = fceConnect(0,"ftp.hiwaay.net","ftp","you@yourisp.com");
```

`BASIC Example`

```
    ' Connect to FTP server
    Code = fceConnect(0,"ftp.hiwaay.net","ftp","you@yourisp.com")
```

ALSO SEE

`fceClose.`

2.7 **fceDelFile** Deletes file from the FTP server.

SYNTAX

```
fceDelFile(Channel, FileName)

  Channel  : (I) Channel number.
  FileName : (P) Name of file to delete.
```

REMARKS

The **fceDelFile** function is used to delete the file 'FileName' from the FTP server.

The delete may fail if either you don't have the necessary permission (as is typical when you connect as an anonymous user) or the file itself is marked as read only.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // delete file
    fceDelFile(0,"PRODUCTS.TXT");
```

BASIC Example

```
    ' delete file
    Code = fceDelFile(0, "PRODUCTS.TXT")
```

ALSO SEE

fcePutFile and fceDelServerDir

2.8 **<u>fceDelServerDir</u>** Deletes the server directory.

SYNTAX

```
fceDelServerDir(Channel, DirName)

  Channel : (I) Channel number.
  DirName : (P) Name of directory to delete.
```

REMARKS

The **fceDelServerDir** function is used to delete the server directory 'DirName' from the FTP server.

The delete may fail if you don't have the necessary permission, as is typical when you connect as an anonymous user.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // delete server directory MYSTUFF.DIR
    Code = fceDelServerDir(0,"MYSTUFF.DIR");
```

BASIC Example

```
    ' delete server directory MYSTUFF.DIR
    Code = fceDelServerDir(0,"MYSTUFF.DIR")
```

ALSO SEE

`fceDelFile`

2.9 **fceDriver** Executes the next state in the FCE state engine.

SYNTAX

```
fceDriver(Channel)

  Channel : (I) Channel number.
```

REMARKS

The **fceDriver** function executes the next state in the FCE state engine.

This function is only used when FCE_SET_AUTO_CALL_DRIVER is set to 0.

Refer to Section 4, "Theory of Operation" in the FCE User's Manual (FCE_USR) for more details.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return = 0 : The driver is finished (idle).
Return > 0 : The driver is not yet finished.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // call driver until it returns 0
    while (fceDriver(0)!=0);
```

BASIC Example

```
    ' call driver until it returns 0
    While fceDriver(0) > 0
      '
    Wend
```

ALSO SEE

See Section 4, "Theory of Operation" in the User's Manual (FCE_USR.PDF). Also view online at http://www.marshallsoft.com/fce_usr.pdf

2.10 <u>**fceErrorText**</u> Formats an error message.

SYNTAX

```
fceErrorText(Channel, ErrCode, Buffer, BufLen)

  Channel : (I) Channel number.
  ErrCode : (I) Error code.
  Buffer  : (P) Pointer to put error message.
  BufLen  : (I) Size of 'Buffer'.
```

REMARKS

The **fceErrorText** function formats the error message for error 'Code' in 'Buffer'.

Call this function when an error (a negative value) is returned from a FCE function so that the error message can be displayed or logged.

RETURNS

```
The number of characters copied to 'Buffer'.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    n = fceErrorText(0,ErrCode,(char *)Buffer,100);
    if(n>0) printf("ERROR %s\n", Buffer);
```

BASIC Example

```
    Buffer = SPACE$(100)
    N = fceErrorText(0, ErrCode, Buffer, 100)
    If N > 0 Then
      Print Buffer
    End If
```

ALSO SEE

```
None.
```

2.11 <u>**fceExtract**</u> Extracts strings from FTP formatted file lists.

SYNTAX

```
fceExtract(Buffer, LineNbr, FieldNbr, BufPtr, BufLen)

  Buffer   : (P) Buffer returned by fceGetList.
  LineNbr  : (I) Line number [1,2,...] wanted.
  FieldNbr : (I) Field number [1,2,...] wanted.
  BufPtr   : (P) Resultant buffer.
  BufLen   : (I) Size of 'BufPtr'.
```

REMARKS

The **fceExtract** function extracts fields from FTP formatted file lists for line 'LineNbr' and field 'FieldNbr'. The extracted substring is copied into 'BufPtr'. Use 'FieldNbr' 0 in order to copy the entire line rather than a field.

A typical line in a full FTP directory listing may look like the following. Note that there are 9 fields.

```
rw rr 1 345 15 100424 Feb 8 16:26 fce4c10b.zip
```

Note that in the line above, field 5 is the file length.

The **fceExtract** function is typically called after calling **fceGetList**.

RETURNS

```
The number of characters copied to 'BufPtr'.
```

EXAMPLES

See the CONFTP example program.

C/C++ Example

```
/* get each field for line 8 (returned from call to fceGetList) */
for(i=1;i<=9;i)
  {Code = fceExtract((char *)DataBuffer, 8, i, (char *)LineBuf, 100);
   printf("FIELD %d: %s \n", i, LineBuf);
  }
```

BASIC Example

```
' get each field for line 8 (returned from call to fceGetList)
For I = 1 To 9
  LineBuf = SPACE$(100)
  Code = fceExtract(DataBuffer, 8, I, LineBuf, 100)
  Print 'FIELD ', I, ' :', LineBuf
Next I
```

ALSO SEE

```
fceGetList
```

2.12 **fceFileLength** Extracts file length from listing field.

SYNTAX

```
fceFileLength (Buffer, FieldBeg, FieldEnd)

  Buffer   : (P) Buffer returned by fceGetList.
  FieldBeg : (I) Field # (1,2,...) to start.
  FieldEnd : (I) Field # (2,3,...) to end.
```

REMARKS

The **fceFileLength** function examines each field in 'Buffer' beginning with field 'FieldBeg' through 'FieldEnd' and returns the value of the first completely numeric field found.

The purpose of this function is to return the value of the file length field. This can be problematic since there is no standard FTP format for file listings. For example, field 6 contains the file length in the first example (from a UNIX server), and field 4 in the second example (from a Windows XP server).

```
rw rr 1 345 15 287967 Feb 8 16:26 fce4pb32.zip

01/06/2003 09:45 AM  287967 fce4pb32.zip
```

RETURNS

The numeric value of the first fully numeric field, or –1 that indicates that no numeric file is found.

EXAMPLES

Note that `fceGetList(0, FCE_FULL_LIST_FILE, . . .)` must be called first, in which the filename wanted in first placed in "DataBuffer". Be sure that 'DataBuffer' is sufficiently large for the full file listing.

C/C++ Example

```
     Code = fceGetList(0, FCE_FULL_LIST_FILE, (char *)DataBuffer, 256);
     . . .
     Value = fceFileLength((char *)DataBuffer,3,7);
     if(Value>=0) printf("Filelength = %d\n",Value);
     else printf("Cannot determine file length\n");
```

BASIC Example

```
     Code = fceGetList(0, FCE_FULL_LIST_FILE, DataBuffer, 256)
     . . .
     Value = fceFileLength(DataBuffer,3,7)
     IF Value >= 0 THEN
       PRINT "Filelength = " + Str$(Value)
     END IF
```

ALSO SEE

fceGetFileSize

2.13 <u>**fceGetDirFiles**</u> Gets (downloads) files from FTP server.

SYNTAX

```
fceGetDirFiles(Channel, Pattern, Buffer, BufLen, CaseSen)

  Channel  : (I) Channel number.
  Pattern  : (P) File pattern of files to be downloaded.
  Buffer   : (P) Work buffer (for file list).
  BufLen   : (I) Size of 'Buffer'.
  CaseSen  : (I) T if pattern is case sensitive.
```

REMARKS

The **fceGetDirFiles** function is used to download all files matching the file pattern 'Pattern' from the FTP server. The 'Pattern' is a filename which may contain '?' and '*' wildcards. The '?' character matches any one character while '*' matches any series of characters. For example, "*.ZIP" specifies all files that end with extension ".ZIP". The 'Buffer' is a work buffer that must be sufficiently large to store all filenames.

Call **fceSetServerDir** to specify the server directory and **fceSetLocalDir** to specify the local directory before downloading.

Note that ASCII transfer mode is normally the default. Call **fceSetMode**(Chan,'B') to set the transfer mode to binary for non-ASCII files.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the mGet example program.

C/C++ Example

```
    char Buffer[64000];
    char *Pattern = "*.txt";
    // download all files matching "*.txt"
    Code = fceGetDirFiles(0, Pattern, (char *)Buffer, 64000, FALSE);
```

BASIC Example

```
    Dim Buffer As String
    Dim Pattern As String
    Buffer = SPACE(64000)
    Pattern = "*.txt"
    // download all files matching "*.txt"
    Code = fceGetDirFiles(0, Pattern, Buffer, 64000, False)
```

ALSO SEE

`fcePutDirFiles`

2.14 <u>**fceGetFile**</u> Gets (downloads) file from FTP server.

SYNTAX

```
fceGetFile(Channel, FileName)

  Channel  : (I) Channel number.
  FileName : (P) Name of file to download.
```

REMARKS

The **fceGetFile** function is used to download the file 'FileName' from the FTP server. The file can be also be renamed when it is saved by specifying "oldname:newname" for filename. See example below.

Call **fceSetServerDir** to specify the server directory and **fceSetLocalDir** to specify the local directory before downloading.

Note that ASCII transfer mode is normally the default. Call **fceSetMode**(Chan,'B') to set the transfer mode to binary for non-ASCII files.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // download file "PRODUCTS.TXT"
    Code = fceGetFile(0,"PRODUCTS.TXT");
    // download "YOURFILE.BIN" and save as "MYFILE.BIN"
    Code = fceGetFile(0, "YOURFILE.BIN:MYFILE.BIN");
```

BASIC Example

```
    ' download file "PRODUCTS.TXT"
    Code = fceGetFile(0,"PRODUCTS.TXT")
    ' download "YOURFILE.BIN" and save as "MYFILE.BIN"
    Code = fceGetFile(0, "YOURFILE.BIN:MYFILE.BIN")
```

ALSO SEE

fcePutFile

2.15 **fceGetFileSize** Gets file size from FTP server.

SYNTAX

```
fceGetFileSize(Channel, FileName)

  Channel  : (I) Channel number.
  FileName : (P) Name of file
```

REMARKS

The **fceGetFileSize** function is used to get the size of file 'FileName' from the FTP server.

The **fceGetFileSize** function uses the "extended FTP" command "SIZE", which is not supported by all FTP servers. In this case, use the **fceFileLength** command instead.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

C/C++ Example

```
    // get size of file "PRODUCTS.TXT"
    Code = fceGetFileSize(0,"PRODUCTS.TXT");
```

BASIC Example

```
    ' get size of file "PRODUCTS.TXT"
    Code = fceGetFileSize(0,"PRODUCTS.TXT")
```

ALSO SEE

```
fceFileLength
```

2.16 <u>**fceGetFileTime**</u> Gets file timestamp from FTP server.

SYNTAX

```
fceGetFile(Channel, FileName, Buffer, BufLen()

  Channel  : (I) Channel number.
  FileName : (P) Name of file
  Buffer   : (P) Buffer into which timestamp is copied
  BufLen   : (I) Length of buffer (should be >= 16)
```

REMARKS

The **fceGetFileTime** function is used to get the timestamp (of last modification) of file 'FileName' from the FTP server. The timestamp should be in GMT (Greenwich Mean Time), although this may vary between individual servers.

The **fceGetFileTime** function uses the "extended FTP" command "MDTM", which is <u>not</u> supported by all FTP servers.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

C/C++ Example

```
    // get timestamp of file "PRODUCTS.TXT"
    Code = fceGetFileTime(0,"PRODUCTS.TXT", (char *)Buffer, 16);
```

BASIC Example

```
    ' get timestamp of file "PRODUCTS.TXT"
    Buffer = SPACE(16)
    Code = fceGetFileTime(0,"PRODUCTS.TXT", Buffer, 16)
```

ALSO SEE

fcePutFile

2.17 **fceGetInteger** Returns numeric parameter for FTP processing.

SYNTAX

```
fceGetInteger(Channel, ParamName)

  Channel   : (I) Channel number.
  ParamName : (I) Parameter name.
```

REMARKS

The **fceGetInteger** function returns the value of the specified parameter 'ParamName'.

Note that the return type is unsigned long.

```
     FCE_GET_BUILD              Returns FCE build number.
     FCE_GET_CONNECT_STATUS     Returns 1 if connected.
     FCE_GET_COUNTER            Returns # times FCE driver was called.
     FCE_GET_FILE_BYTES_RCVD    Returns # file bytes received.
     FCE_GET_FILE_BYTES_SENT    Returns # file bytes sent.
     FCE_GET_RESPONSE           Returns last (numerical) FTP response.
     FCE_GET_SOCKET             Returns control socket number.
     FCE_GET_SOCK_ERROR         Returns last socket error code.
     FCE_GET_TOTAL_BYTES_RCVD   Returns total bytes received.
     FCE_GET_TOTAL_BYTES_SENT   Returns total file bytes sent.
     FCE_GET_VERSION            Returns FCE version.
     FCE_GET_QUEUE_ZERO         Returns # times fceQueueLoad returns 0.
     FCE_GET_DATA_PORT          Returns last data port used.
     FCE_GET_DAYS_LEFT          Returns # days left for evaluation version.
     FCE_SKEY_WAS_SEEN          Returns 1 if S/KEY was seen while connecting.
```

RETURNS

```
Value of parameter requested [long integer (L)].
```

EXAMPLES

Most example programs call `fceGetInteger`.

`C/C++ Example`

```
    // display FCE version and build number.
    Version = fceGetInteger(0, FCE_GET_VERSION);
    printf("FCE32 Version: %1d.%1d.%1d \n",
      0x0f&(Version>>8),0x0f&(Version>>4),0x0f&Version);
```

`BASIC Example`

```
    Version = fceGetInteger(0, FCE_GET_VERSION)
    S = Hex$(Version)
    Print Mid$(S, 1, 1) + "." + Mid$(S, 2, 1) + "." + Mid$(S, 3, 1)
```

ALSO SEE

```
fceGetString
```

2.18 <u>**fceGetList**</u> Gets file list from FTP server.

SYNTAX

```
fceGetList(Channel, Flag, Buffer, BufLen)

  Channel : (I) Channel number.
  Flag    : (I) Listing type flag (see below).
  Buffer  : (P) List buffer.
  BufLen  : (I) Size of 'Buffer'
```

REMARKS

The **fceGetList** function downloads the directory list from the FTP server.

If 'FCE_FULL_LIST' is passed for 'Flag', a full directory listing is returned in 'Buffer'. Note that the exact format of the list depends on the particular FTP server.

If 'FCE_NAME_LIST' is passed for 'Flag', a listing is returned consisting of file names only. Note that some FTP servers do not support the name list function.

If 'FCE_FULL_LIST_FILE' is passed for 'Flag', the filename to list is taken from 'Buffer'. If the file exists, a listing of this file is returned.

If 'FCE_NAME LIST_FILE' is passed for 'Flag', the filename to list is taken from 'Buffer'. If the file exists, the name of this file is returned. Be sure to check the return code length.

File lists consist of a zero terminated list of file entries, each of which is terminated by a carriage return, line feed pair. Also check the return code, which contains the length of the characters placed in 'Buffer'.

Note: The buffer passed to **fceGetList** <u>must</u> have space for 'BufLen' bytes.

RETURNS

```
Return < 0 : An error has occurred (buffer overflow). Call fceErrorText.
Return > 0 : Number of characters copied to 'Buffer'.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // get file name list
    Code = fceGetList(0, FCE_NAME_LIST, (char *)Buffer, 2000);
    if(Code>0) printf("%s", Buffer);
```

BASIC Example

```
    ' get file name list
    Buffer = SPACE$(2000)
    Code = fceGetList(0, FCE_NAME_LIST, Buffer, 2000)
    If Code > 0 Then
      Print Buffer
    End If
```

2.19 <u>**fceGetLocalDir**</u> Returns the local upload/download directory.

SYNTAX

```
fceGetLocalDir(Channel, Buffer, BufLen)

  Channel : (I) Channel number.
  Buffer  : (P) String buffer.
  BufLen  : (I) Size of 'Buffer'.
```

REMARKS

The **fceGetLocalDir** function returns the local upload/download directory.

The local upload/download directory is the directory used for all uploads and downloads. The default is the current directory (".").

Both relative and absolute directories may be specified.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return > 0 : The number of characters copied.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    char Buffer(128);
    Get local upload/download directory.
    fceGetLocalDir(0, (char *)Buffer, 128);
```

BASIC Example

```
    Dim Buffer As String * 128
    ' Get local upload/download directory.
    fceGetLocalDir(0, Buffer, 128)
```

ALSO SEE

fceSetLocalDir

2.20 <u>**fceGetLocalFList**</u> Gets list of all files in local directory.

SYNTAX

```
fceGetLocalFList(Channel, Buffer, BufLen)

  Channel : (I) Channel number.
  Buffer  : (P) String buffer.
  BufLen  : (I) Size of 'Buffer'.
```

REMARKS

The **fceGetLocalFList** function is used to return a list of files in the local upload/download directory.

Note that the local upload/download directory is set with **fceSetLocalDir** and read by **fceGetLocalDir**.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return = 0 : No files in local directory.
Return > 0 : The number of filenames in 'Buffer'.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
     int FileCount;
     FileCount = fceGetLocalFList(0, (char *)Buffer, MAX_BUF);
```

BASIC Example

```
     Dim FileCount As Integer
     Dim Buffer As String * 5001
     Buffer = Space(5001)
     FileCount = fceGetLocalFList(0, Buffer, 5000)
```

ALSO SEE

fceGetLocalFSize

2.21 **fceGetLocalFSize** Gets size of file in upload/download directory.

SYNTAX

```
fceGetLocalFSize (Channel, FileName)

  Channel  : (I) Channel number.
  FileName : (P) Name of file in local directory.
```

REMARKS

The **fceGetLocalFSize** function is used to return the length of the file in the local upload/download directory specified by 'FileName'.

Note that the local upload/download directory is set with **fceSetLocalDir** and read by **fceGetLocalDir**.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return > 0 : File length of 'FileName'.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
     int FileCount;
     FileCount = fceGetLocalFSize (0, (char *)"MyFile.bin");
```

BASIC Example

```
     Dim FileCount As Integer
     Dim FileName As String
     FileName = "MyFile.bin"
     FileCount = fceGetLocalFSize(0, FileName)
```

ALSO SEE

fceGetLocalFList

2.22 <u>**fceGetServerDir**</u> Returns the FTP server directory.

SYNTAX

```
fceGetServerDir(Channel, Buffer, Buflen)

  Channel : (I) Channel number
  Buffer  : (P) String buffer.
  BufLen  : (I) Size of 'Buffer'.
```

REMARKS

The **fceGetServerDir** function returns the FTP server directory.

Note that most FTP servers will restrict clients as to which directories on the server can be accessed.

The default is the current logged directory on the FTP server.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return > 0 : The number of characters copied.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // copy directory string to 'Buffer'
    Code = fceGetServerDir(0, (char *)Buffer, 65);
    printf("Server directory is %s\n", Buffer);
```

BASIC Example

```
    ' copy directory string to 'Buffer'
    Buffer = SPSACE$(65)
    Code = fceGetServerDir(0, Buffer, 65)
    Print "Server directory is ", Buffer
```

ALSO SEE

fceSetServerDir

2.23 **fceGetString** Returns string parameter for FTP processing.

SYNTAX

```
fceGetString(Channel, ParamName, Buffer, BufLen)

  Channel   : (I) Channel number
  ParamName : (P) Parameter name
  Buffer    : (P) String buffer.
  BufLen    : (I) Size of 'Buffer'.
```

REMARKS

The **fceGetString** function returns the string parameter 'ParamName'.

```
     FCE_GET_LINE_COUNT      Returns the # lines in 'Buffer'.
     FCE_GET_LAST_RESPONSE   Returns last FTP response.
     FCE_GET_REGISTRATION    Returns registration string.
     FCE_GET_SERVER_IP       Returns IP address of FTP server.
     FCE_GET_LOCAL_IP        Returns local IP address.
     FCE_GET_FULL_RESPONSE   Returns multi-line server response.
     FCE_GET_REGISTRATION    Returns registration string.
     FCE_GET_LAST_RESPONSE   Returns last server response.
     FCE_GET_SERVER_IP       Returns IP address of server.
     FCE_GET_LINE_COUNT      Returns # lines in 'Buffer'.
     FCE_GET_LOCAL_IP        Returns local IP address (once connected)
     FCE_GET_ERROR_LINE      Returns text of error from last server response.
```

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return > 0 : Number of characters copied to 'Buffer', or(FCE_GET_LINE_COUNT)
             the number of lines in 'Buffer'.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // display registration string within the DLL
    Code = fceGetString(0, FCE_GET_REGISTRATION, (char *)Buffer, 50);
    printf("Registration = '%s'\n", Buffer);
```

BASIC Example

```
    ' display registration string within the DLL
    Buffer = SPACE$(50)
    Code = fceGetString(0, FCE_GET_REGISTRATION, Buffer, 50)
    Print "Registration ", Buffer
```

ALSO SEE

`fceGetInteger`

2.24 <u>**fceGetSubDirs**</u> Returns List of Sub-directories

SYNTAX

```
fceGetSubDirs(Channel, ListBuf, ListLen, Buffer, BufLen, Flags)
  Channel   : (I) Channel number
  ListBuf   : (P) List buffer (work buffer).
  ListLen   : (I) Size of 'ListBuf'.
  Buffer    : (P) String buffer.
  BufLen    : (I) Size of 'Buffer'.
  Flags     : (I) Flags (reserved)
```

REMARKS

The **fceGetSubDirs** function copies the list of all sub-directory filenames in the current server directory to 'Buffer'. Each filename copied into 'Buffer' is terminated by a CR/LF pair.

The 'ListBuf' buffer must be large enough to hold all filenames in the current server directory.

RETURNS

```
The # characters copied into 'Buffer'.
```

EXAMPLES

C/C++ Example

```
    char Work[65535];
    char Buff[4096];
    Code = fceGetSubDirs(0, (char *)Work, 65536, (char *)Buff, 4096, 0);
```

BASIC Example

```
    DIM Work As String
    DIM Buffer As String
    Work = SPACE(65535)
    Buff = SPACE(4096)
    Code = fceGetSubDirs(0, Work, 65536, Buff, 4096, 0)
```

ALSO SEE

```
None.
```

2.25 <u>**fceGetTicks**</u> Returns # milliseconds since system boot.

SYNTAX

```
fceGetTicks()
```

REMARKS

The **fceGetTicks** function returns the system time in milliseconds since the system was booted. **fceGetTicks** calls the Windows API function GetCurrentTime. This function is provided as a convenience for computer languages in which GetCurrentTime can not be called directly.

RETURNS

```
The system time in milliseconds.
```

EXAMPLES

C/C++ Example

```
    ULONG TimeMark;
    TimeMark = fceGetTicks();
    printf("Time is %ld ticks\n", TimeMark);
```

BASIC Example

```
    DIM TimeMark As LONG
    TimeMark = fceGetTicks()
    Print "Time is " + Str$(TimeMark)
```

ALSO SEE

```
None.
```

2.26 **fceHello** Issues NOOP command to server.

SYNTAX

```
fceHello(Channel)

  Channel : (I) Channel number.
```

REMARKS

The **fceHello** function issues a "NOOP" command to the server. The primary purpose for this command is to determine if the server is still responding to commands.

This function can sometimes be used as a "keep alive" command, although most servers will drop your connection after a fixed period of time unless data is transferred.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
Return > 0 : The number of characters copied.
```

EXAMPLES

C/C++ Example

```
    // is the server responding ?
    Code = fceHello(0);
    if(Code>=0) printf("Server is responding\n");
```

BASIC Example

```
    ' is the server responding ?
    Code = fceHello(0)
    If Code >=0 Then
      Print "Server is responding"
    End If
```

ALSO SEE

None.

2.27 **fceIsConnected** Returns the current connection status.

SYNTAX

```
fceIsConnected(Channel)

  Channel : (I) Channel number.
```

REMARKS

The **fceIsConnected** function is used determine the current connection status. It returns TRUE for a live connection and FALSE if the connection has been dropped.

EXAMPLE (C/C++)

```
     //test connection
     if(!fceIsConnected(vSock))
       {printf("*** ERROR: Connection has been dropped!\n");
        break;
       }
```

EXAMPLE (VB)

```
      Dim vSock As Long
      If fceIsConnected(vSock) = 0 Then
         Result.Text = "*** ERROR: Connection has been dropped!"
      End If
```

RETURNS

```
True  : Connective is OK.
False : Connection has been dropped.
```

2.28 <u>**fceMakeServerDir**</u> Creates server directory.

SYNTAX

```
fceMakeServerDir(Channel, DirName)

  Channel : (I) Channel number.
  DirName : (P) Name of directory to make.
```

REMARKS

The **fceMakeServerDir** function is used to make (create) server directory 'DirName' on the FTP server.

The make may fail if you don't have the necessary permission, as is typical when you connect as an anonymous user.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // create new directory
    Code = fceMakeServerDir(0, "MYSTUFF.DIR");
```

BASIC Example

```
    ' create new directory
    Code = fceMakeServerDir(0, "MYSTUFF.DIR")
```

ALSO SEE

`fceDelServerDir`

2.29 **fceMatchFile:** Match next file name in list.

SYNTAX

```
fceMatchFile(ListBuf,Start,NameBuf,NameLen,FileSpec,CaseFlag)

  ListBuf  : (P) Multi-line filename buffer.
  Start    : (I) Offset into above to start.
  NameBuf  : (P) Buffer to put matched name into.
  NameLen  : (I) Size of above.
  FileSpec : (P) File specification pattern.
  CaseFlag : (I) Case sensitive comparisons if true.
```

REMARKS

The **fceMatchFile** function is used to copy the next filename into 'NameBuf' from the 'ListBuf' starting at byte offset 'Start' that matches the file specification pattern 'FileSpec'.

'ListBuf' must consist of one or more filenames separated by carriage return, line feed pairs. This is normally returned by the FTP server when requesting a name list (FCE_NAME_LIST).

The 'FileSpec' is a filename which may contain '?' and '*' wildcards. The '?' character matches any one character while '*' matches any series of characters. For example, "*.ZIP" specifies all files that end with extension ".ZIP".

fceMatchFile returns the offset to the next file name after the matched file. Pass this offset as the 'Start' parameter in the next call to **fceMatchFile** in order to find the next matching file name.

The primary purpose of **fceMatch file** is to enable multi-file transfers based on a filename pattern. See the MGET example program for a complete example.

RETURNS

```
Return > 0 : The offset to the next file name in 'ListBuf' after matched file.
Return = 0 : The end of the list has been reached.
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the MGET example program.

ALSO SEE

```
fceExtract
```

2.30 <u>**fcePutDirFiles**</u> Puts (uploads) files to FTP server.

SYNTAX

```
fcePutDirFiles(Channel, Pattern, Buffer, BufLen, CaseSen)

  Channel  : (I) Channel number.
  Pattern  : (P) File pattern of files to be downloaded.
  Buffer   : (P) Work buffer (for file list).
  BufLen   : (I) Size of 'Buffer'.
  CaseSen  : (I) T if pattern is case sensitive.
```

REMARKS

The **fcePutDirFiles** function is used to upload all files matching the file pattern 'Pattern' to the FTP server. The 'Pattern' is a filename which may contain '?' and '*' wildcards. The '?' character matches any one character while '*' matches any series of characters. For example,"*.ZIP" specifies all files that end with extension ".ZIP". The 'Buffer' is a work buffer that must be sufficiently large to store all filenames.

Call **fceSetServerDir** to specify the server directory and **fceSetLocalDir** to specify the local directory before uploading.

Note that ASCII transfer mode is normally the default. Call **fceSetMode**(Chan,'B') to set the transfer mode to binary for non-ASCII files.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the mPut example program.

C/C++ Example

```
    char Buffer[64000];
    char *Pattern = "*.txt";
    // upload all files matching "*.txt" (in the local directory)
    Code = fcePutDirFiles(0, Pattern, (char *)Buffer, 64000, FALSE);
```

BASIC Example

```
    Dim Buffer As String
    Dim Pattern As String
    Buffer = SPACE(64000)
    Pattern = "*.txt"
    // upload all files matching "*.txt"
    Code = fcePutDirFiles(0, Pattern, Buffer, 64000, False)
```

ALSO SEE

```
fceGetDirFiles
```

2.31 **fcePutFile** Uploads file to FTP server.

SYNTAX

```
fcePutFile(Channel, FileName)

  Channel  : (I) Channel number.
  FileName : (P) Name of file to upload.
```

REMARKS

The **fcePutFile** function uploads the file 'FileName' to the FTP server.

Call **fceSetServerDir** to specify the server directory and **fceSetLocalDir** to specify the local directory before uploading.

The file 'FileName' to be uploaded must be in the local upload/download directory. Transfer mode is by default ASCII. For binary mode, pass ‘B’ to **fceSetMode** before calling **fcePutFile.**

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // upload file
    Code = fcePutFile(0, "COMMENTS.TXT");
```

BASIC Example

```
    ' upload file
    Code = fcePutFile(0, "COMMENTS.TXT")
```

ALSO SEE

```
fceGetFile.
```

2.32 **fceRelease** Releases FCE.

SYNTAX

```
fceRelease
```

REMARKS

The **fceRelease** function releases the FCE system. This should be the very last function called.

fceClose should be called for all channels before calling **fceRelease**.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

All example programs call `fceRelease`.

C/C++ Example

```
    // Terminate FCE
    fceRelease();
```

BASIC Example

```
    ' Terminate FCE
    Code = fceRelease()
```

ALSO SEE

```
fceAttach.
```

2.33 **fceSetInteger** Sets numeric parameter for FTP processing.

SYNTAX

```
fceSetInteger(Channel, ParamName, ParamValue)

  Channel    : (I) Channel number.
  ParamName  : (I) Parameter name.
  ParamValue : (L) Parameter value.
```

REMARKS

The **fceSetInteger** function sets the numeric parameter 'ParamName' to the value 'ParamValue'.

Parameter	Name Default
FCE_SET_AUTO_CALL_DRIVER	1 (TRUE)
FCE_SET_CLOSE_LINGER	50
FCE_SET_CONNECT_WAIT	60000
FCE_SET_DATA_PORT	(none)
FCE_SET_FTP_PORT	21
FCE_SET_MAX_LINE_WAIT	20000
FCE_SET_MAX_LISTEN_WAIT	25000
FCE_SET_MAX_RESPONSE_WAIT	10000
FCE_SET_MIN_LINE_WAIT	0
FCE_SET_MIN_RESPONSE_WAIT	0
FCE_SET_PASSIVE	0 (FALSE)
FCE_SET_SLEEP_TIME	20
FCE_SET_WRITE_BUFSIZE	65536
FCE_SET_MASTER_INDEX	0
FCE_SET_APPEND_MODE	0 (FALSE)
FCE_SET_CLIENT_OFFSET	0
FCE_SET_SERVER_OFFSET	0
FCE_SET_BLOCKING_MODE	1 (TRUE)
FCE_HIDE_PASSWORD	0 (FALSE)
FCE_SET_FIRST_DATA_PORT	Depends on # channels.
FCE_SET_LAST_DATA_PORT	Depends on # channels.
FCE_CLOSE_LOG_FILE	None.
FCE_AUTO_LOG_CLOSE	0 (FALSE)
FCE_STATUS_BEFORE_WRITE	1 (TRUE)
FCE_LOCAL_DIR_IS_CDROM	0 (FALSE)
FCE_DISABLE_SKEY	0 (FALSE)

`FCE_SET_AUTO_CALL_DRIVER` enables and disables automatic calling of **fceDriver**.

`FCE_SET_CLOSE_LINGER` is the "linger" time after an upload is completed before closing the data socket. Setting this value too small causes the data socket to be closed before the last block of data is transmitted.

`FCE_SET_CONNECT_WAIT` is the maximum time allowed to complete a connection to the FTP server.

`FCE_SET_DATA_PORT` specifies the port number to use (in non-passive mode) for the next list or file transfer command.

`FCE_SET_FTP_PORT` is the port number to use when connecting to the FTP server. The default is the well-known port number 21.

`FCE_SET_MAX_LINE_WAIT` is the time after which a "time out" error is declared if the server has not responded.

`FCE_SET_MAX_LISTEN_WAIT` is the time after which a "time out" error is declared while waiting for a data port "Listen" to complete.

`FCE_SET_MAX_RESPONSE_WAIT` is the time after which a "time out" error occurs if the server has not responded.

`FCE_SET_MIN_LINE_WAIT` is the delay before checking if the server is ready to accept the next line of input.

`FCE_SET_MIN_RESPONSE_WAIT` is the delay before looking for the server's response.

`FCE_HIDE_PASSWORD` is used to direct FCE to replace the password characters with asterisks in the in log file. Pass 1 to hide your password and 0 to allow the password in the log file. The default is 0; passwords are not "hidden".

`FCE_SET_FIRST_DATA_PORT` specifies the first data port to be used in the allowed port range for file transfers (list, uploads, and downloads). This is useful when a range of ports that are allowed through a firewall must be specified.

`FCE_SET_LAST_DATA_PORT` specifies the last data port to be used in the allowed port range for file transfers (list, uploads, and downloads). This is useful when a range of ports that are allowed through a firewall must be specified.

`FCE_CLOSE_LOG_FILE` is used to close the log file immediately.

`FCE_AUTO_LOG_CLOSE` specifies that the log file should be closed automatically whenever fceClose is called. The default value is 1 (TRUE). Pass 0 to keep the log file open when fceClose is called.

`FCE_SET_PASSIVE` sets passive mode on (1) and off (0). Passive mode means that the server specifies the data port rather than the client when listing or transferring files.

`FCE_SET_SLEEP_TIME` is the sleep time (in milliseconds) when waiting for socket I/O to complete. Useful in multi threaded environments.

`FCE_SET_WRITE_BUFSIZE` is the transmit block size. The maximum value is 65536 (64KB). Note that some FTP servers can not handle high upload rates

`FCE_SET_MASTER_INDEX` is the last index (into the internal Winsock IP address table) searched when calling **fceGetServerIP**. This applies ONLY to multi-homed (multiple IP addresses) local machines.

`FCE_SET_SERVER_OFFSET` sets the server file offset for the next call to **fceGetFile**. This allows an interrupted download to be resumed. `FCE_APPEND_MODE` must also be set for the offset value to be used. Refer to `FCE_SET_APPEND_MODE` below.

`FCE_SET_CLIENT_OFFSET` sets the client file offset for the next call to **fceGetFile** or **fcePutFile**. This allows an interrupted upload or download to be resumed. `FCE_APPEND_MODE` must also be set for the offset value to be used. Refer to `FCE_SET_APPEND_MODE` below.

`FCE_SET_APPEND_MODE` sets the upload/download mode to "append". The next file uploaded (with `fcePutFile`) or downloaded (with `fceGetFile`) will be appended to the existing file. Append mode stays in effect for the next upload or download only. For more information, refer to section "Using Append Mode for Uploads" and "Using Append Mode for Downloads" in the User's Manual (FCE_USR). Also view online at `http://www.marshallsoft.com/fce_usr.pdf`

`FCE_SET_BLOCKING_MODE` sets the blocking mode used when connecting. Pass TRUE (default) to enable blocking while connecting, and FALSE (0) to disable blocking mode while connecting.

`FCE_STATUS_BEFORE_WRITE` if set to true, causes the WRITE status to always be checked before writing.

`FCE_LOCAL_DIR_IS_CDROM` allows the local directory to be a read-only device such as a CDROM.

`FCE_DISABLE_SKEY` disables S/KEY processing.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

Most example programs call fceSetInteger.

`C/C++ Example`

```
    // disable the automatic calling of the state driver.
    fceSetInteger(0, FCE_SET_AUTO_CALL_DRIVER, 0);
```

`BASIC Example`

```
    ' disable the automatic calling of the state driver.
    Code = fceSetInteger(0, FCE_SET_AUTO_CALL_DRIVER, 0)
```

ALSO SEE

`fceSetString`

2.34 <u>**fceSetLocalDir**</u> Sets the local upload/download directory.

SYNTAX

```
fceSetLocalDir(Channel, DirName)

  Channel : (I) Channel number.
  DirName : (P) Local directory path.
```

REMARKS

The **fceSetLocalDir** function sets the local computer upload/download directory. The upload/download directory is the directory used by FCE for all uploads and downloads.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // specify the local upload/download directory.
    fceSetLocalDir(0, "C:\\TEMP");
```

BASIC Example

```
    ' specify the local upload/download directory.
    Code = fceSetLocalDir(0, "C:\TEMP");
```

ALSO SEE

```
fceGetLocalDir
```

2.35 **fceSetMode** Sets FTP transfer mode.

SYNTAX

```
fceSetMode(Channel, Mode)

  Channel : (I) Channel number.
  Mode    : (I) transfer mode ('A' or 'B').
```

REMARKS

The **fceSetMode** function sets the FTP transfer mode. Pass 'A' to specify ASCII mode and 'B' to specify binary mode.

Since the FTP default is usually ASCII, it is good practice to always specify the transfer mode before the first call to **fceGetFile** or **fcePutFile**.

If unsure of the transfer mode, choose binary.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // set binary mode
    fceSetMode(0, 'B');
```

BASIC Example

```
    ' set binary mode
    Code = fceSetMode(0, ASC("B"))
```

ALSO SEE

```
fceGetFile and fcePutFile.
```

2.36 <u>**fceSetServerDir**</u> Sets the remote FTP directory .

SYNTAX

```
fceSetServerDir(Channel, DirName)

  Channel : (I) Channel number.
  DirName : (P) Directory name.
```

REMARKS

The **fceSetServerDir** sets the FTP directory to 'DirName' that is used for subsequent FCE calls.

Note that UNIX FTP servers use forward slashes for directories while Windows FTP servers use backward slashes.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // note forward slashes
    Code = fceSetServerDir (0, "marshallsoft/other")
```

BASIC Example

```
    ' note forward slashes
    Code = fceSetServerDir (0, "marshallsoft/other")
```

ALSO SEE

None.

2.37 **fceSetString** Sets string parameter for FTP processing.

SYNTAX

```
fceSetString(Channel, ParamName, ParamPtr)

  Channel   : (I) Channel number.
  ParamName : (I) Parameter name.
  ParamPtr  : (P) Parameter string.
```

REMARKS

The **fceSetString** function sets the string parameter 'ParamName' to 'ParamPtr'.

`FCE_SET_LOG_FILE` is used to specify the log file name. Log files can be quite large, so use only when necessary.

`FCE_WRITE_TO_LOG` is used to write a string (message) to log file.

`FCE_BIND_TO_LOCAL_IP` is used to bind the control port to the specified local IP address.

RETURNS

```
Return < 0 : An error has occurred. Call fceErrorText.
```

EXAMPLES

See the WINFTP example program.

C/C++ Example

```
    // open LOG file
    fceSetString(0, FCE_SET_LOG_FILE, "program.log");
```

BASIC Example

```
    ' open LOG file
    Code = fceSetString(0, FCE_SET_LOG_FILE, "program.log")
```

ALSO SEE

`fceSetInteger`

2.38 **fceShortToByte** Converts 16-bit ASCII character buffer to 8-bit

SYNTAX

```
fceShortToByte(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **fceShortToByte** function converts the (null terminated) character buffer ‘Buffer’ from 16-bit Unicode ASCII characters to 8-bit ASCII characters.

The buffer must be null terminated (last character is a hex 00).

This function is only necessary when working with 16-bit Unicode ASCII characters in C# and Delphi 2005.

RETURNS

```
None.
```

EXAMPLES

See C# example cs_get.csproj

C# Example

```
     NameString = “MyFile.zip\0”
     char[] NameBuffer = NameString.ToCharArray();
     // convert (null terminated) 16-unicode buffer to 8-bit
     fixed (char* pNameBuffer = NameBuffer)
     fceShortToByte(pNameBuffer);
```

ALSO SEE

```
fceByteToShort
```

2.39 <u>**fceToInteger**</u> Converts ASCII text to integer

SYNTAX

```
fceToInteger(Buffer)

  Buffer : (P) text buffer containing ASCII digits
  Start  : (I) offset to start of first digit
  Count  : (I) maximum number of characters to convert
```

REMARKS

The **fceToInteger** function provides a convenient way to convert text to an integer. For example, if the text buffer passed to **fceToInteger** contains “ABC123XYZ”, calling **fceToInteger**(Buffer, 3, 3) will return the integer 123. The first character that is not a (decimal) digit will terminate the conversion, so **fceToInteger**(Buffer, 3, 8) will also return 123 but **fceToInteger**(Buffer, 3, 2) will return 12.

The buffer <u>must</u> be null terminated (last character is a hex 00).

RETURNS

The converted integer. Zero is returned if no integer digits are found.

EXAMPLES

See the MDTM example program in the APPS directory.

3. FCE Error Return Code List

The complete list of FCE error codes follows.

FCE_ABORTED	Internal checksum fails!
FCE_ACCEPT_SILENT	Timed out waiting for accept.
FCE_ALREADY_ATTACHED	Already attached.
FCE_BAD_STATUS_FLAG	Bad status flag passed to fceStatus.
FCE_BUFFER_OVERFLOW	List buffer overflow.
FCE_CANNOT_ALLOC	Cannot allocate memory.
FCE_CANNOT_COMPLY	Cannot comply.
FCE_CANNOT_CREATE_SOCK	Cannot create socket.
FCE_CANNOT_OPEN	Cannot open file.
FCE_CHAN_OUT_OF_RANGE	Channel out of range.
FCE_CONNECT_ERROR	Error attempting to connect.
FCE_EOF	Socket has been closed.
FCE_FILE_IO_ERROR	File I/O error.
FCE_INVALID_SOCKET	Invalid socket.
FCE_IS_BLOCKING	WINSOCK is currently blocking.
FCE_LISTEN_ERROR	Listen error.
FCE_LISTENER_SILENT	No response on listener socket.
FCE_MODE_NOT_AB	Must specify 'A' or 'B' for mode.
FCE_NO_GREETING	Missing server greeting message.
FCE_NO_HOST	No host name.
FCE_NO_SERVER	Cannot find FTP server.
FCE_NO_SOCK_ADDR	No available sockaddr structures.
FCE_NOT_ATTACHED	Must call fceAttach first.
FCE_NOT_COMPLETED	LIST/GET/PUT not completed.
FCE_NOT_SERVER	Illegal chars in server name.
FCE_PASS_NULL_ARG	PASSWORD not specified.
FCE_PASV_ERROR	Cannot find PASV port.
FCE_PORT_RANGE	Port number out of range.
FCE_SERVER_ERROR	FTP server returned error.
FCE_SERVER_NULL_ARG	SERVER not specified.
FCE_SOCK_READ_ERROR	Socket read error.
FCE_SOCK_WRITE_ERROR	Socket write error.
FCE_TIMED_OUT	Socket timed out.
FCE_USER_NULL_ARG	USER name not specified.
FCE_NOT_CONNECTED	Not connected to server.

The numerical value for each error codes is listed in the file **fceErrors.txt**.