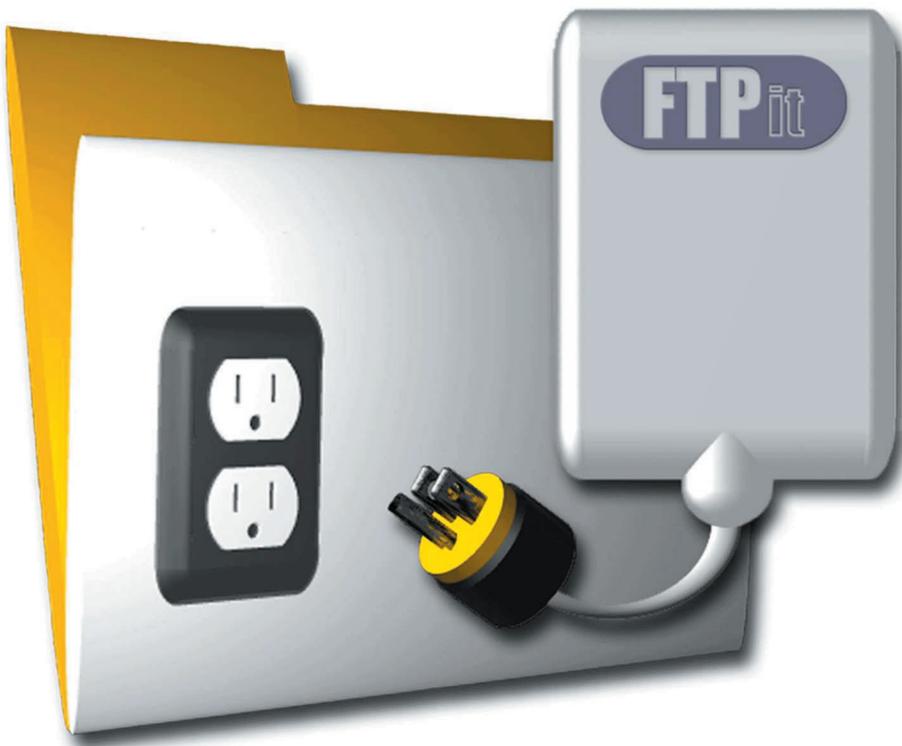




**Comm-Unity
Networking
Systems**



FTPit

A Plug-in for FileMaker Pro

- Upload or download any file you have access to on any FTP site
- Catalog all the files on an FTP site using FTPit's directory listing functions
- Watch everything FTPit is doing with its Status Window
- Queue multiple files and move on to something else while FTPit transfers the files in the background.

Comm-Unity Networking Systems
8652 Camp Bowie West
Fort Worth, Texas 76116

<http://www.cnsplug-ins.com/>
FTPit@comm-unity.net

Phone: 817-560-4226
Fax: 817-244-0340

Contents

Introduction.....	5
Features	5
Installation & Configuration.....	6
Installation.....	6
Basics Tab.....	8
Advanced Tab.....	9
About Tab.....	9
Registration.....	10
Working with FTPit.....	10
Text Transfer Example Explained.....	12
Text Upload	12
Upload Script.....	12
Text Download	14
Single File Transfer Example Explained	16
Upload Script.....	16
Download Script.....	18
Multiple File Transfer Example Explained	18
Multi Upload Script	19
Multi Download Script	21
FTPit External Function Reference	22
Available Functions	22
doFTP-Append	22
doFTP-ChangeDir	22
doFTP-ChangeDirUp.....	23
doFTP-CompletedScript.....	23
doFTP-Connect	24
doFTP-CurrentDir.....	24
doFTP-DataPortRangeEnd.....	24
doFTP-DataPortRangeStart.....	25
doFTP-Delete.....	25
doFTP-Disconnect	25

<i>doFTP-Get</i>	25
<i>doFTP-GetLocalRoot</i>	26
<i>doFTP-GetPathToDB</i>	26
<i>doFTP-GetPathToFM</i>	27
<i>doFTP-GetQueue</i>	27
<i>doFTP-GetText</i>	28
<i>doFTP-HideStatus</i>	28
<i>doFTP-Host</i>	28
<i>doFTP-IsConnected</i>	29
<i>doFTP-LastCompleted</i>	29
<i>doFTP-ListFirst</i>	30
<i>doFTP-ListNext</i>	30
<i>doFTP-LocalChangeDir</i>	31
<i>doFTP-LocalChangeDirUp</i>	31
<i>doFTP-LocalCurrentDir</i>	31
<i>doFTP-LocalDelete</i>	32
<i>doFTP-LocalListFirst</i>	32
<i>doFTP-LocalListNext</i>	32
<i>doFTP-LocalMakeDir</i>	33
<i>doFTP-LocalRemoveDir</i>	33
<i>doFTP-LocalRename</i>	33
<i>doFTP-LocalSize</i>	34
<i>doFTP-MakeDir</i>	34
<i>doFTP-MoveStatus</i>	34
<i>doFTP-Passive</i>	35
<i>doFTP-Password</i>	35
<i>doFTP-Port</i>	35
<i>doFTP-Put</i>	36
<i>doFTP-PutText</i>	36
<i>doFTP-Quote</i>	37
<i>doFTP-Register</i>	37
<i>doFTP-RemoveDir</i>	37
<i>doFTP-Rename</i>	38
<i>doFTP-ShowStatus</i>	38
<i>doFTP-Size</i>	39
<i>doFTP-TextAcquire</i>	39
<i>doFTP-TextAppend</i>	40
<i>doFTP-TextAssign</i>	40
<i>doFTP-Type</i>	41
<i>doFTP-Username</i>	41
<i>doFTP-Version</i>	42
<i>doFTP-WaitForQueue</i>	42
 Understanding Error Responses	 44

Credits & Contact Information 45

Index of Figures

Figure 1. Installing on Macintosh 6
Figure 2. Installing on Windows 7
Figure 3. Configuration Dialog Basics..... 8
Figure 4. Configuration Dialog About..... 9

Introduction

FTPit is an exciting plug-in brought to you from the makers of the leading email plug-ins, SMTPit and POP3it. FTPit uses the File Transfer Protocol (FTP) to allow you to transfer files to and from any FTP site. FTPit also allows you to upload text from a FileMaker field to a text file on the FTP site or download a text file directly into a field in your FileMaker database. This can be used in a solution that creates static web pages for a web site. Instead of launching a separate FTP client to upload the html files your solution creates, you can do it all from within FileMaker Pro with the click of a button.

Features

- Upload or download any file you have access to on any FTP site.
- Upload a complete website using only FileMaker Pro and FTPit.
- Queue multiple files and move on to something else while FTPit transfers the files in the background.
- Watch everything that FTPit is doing with it's Status Window.
- Catalog all the files on an FTP site using FTPit's directory listing functions.
- NEW! - Download text files directly into a field in your database.
- NEW! - More example databases show you how to transfer single or multiple files and how to use the Text functions.
- NEW! - New WaitForQueue function allows your complex FTPit scripts to flow more naturally.

Installation & Configuration

Installation

Installing FTPit is very easy. If FileMaker Pro is open, then close it. Next, locate the folder on your hard drive that contains your FileMaker Pro application. For Macintosh, this is usually in the Applications folder; while on Windows, it is usually in the Program Files folder. Next to the FileMaker Pro application, you should see a folder named FileMaker Extensions (on Macintosh; See Figure 1) or System (on Windows; See Figure 2). Copy the plug-in file into this folder to install it into FileMaker Pro.



Figure 1. Installing on Macintosh



Figure 2. Installing on Windows

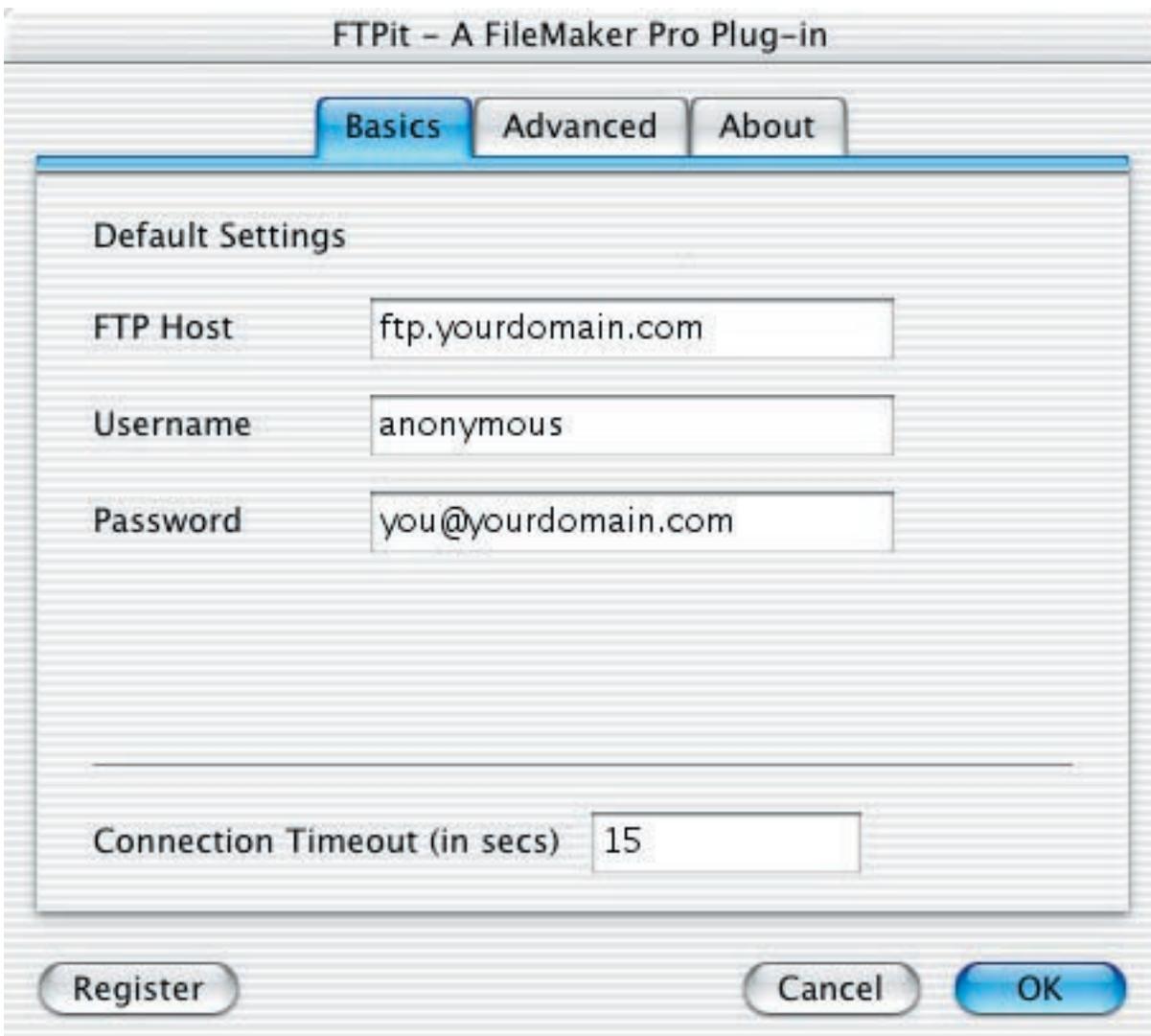
After installing the plug-in as described above, you can open FileMaker Pro again and set the default preferences if you wish. To do this, go to Edit, Preferences, Application, click on the Plug-ins tab, and double-click the FTPit plug-in.

Basics Tab

Once the Configuration Dialog is open, click the **Basics** tab (See Figure 1) where you can set the following default values. **FTP Host** is where you enter the domain name or IP address of the FTP site you want to connect to. **Username** is where you enter the default username of the FTP account that you want to connect to. **Password** is where you enter the default password for the FTP account you want to connect to.

FTPit will refer to the default settings when you do not specifically set values in your scripts. In other words, if you do not define a username in your FTPit related script (with the [doFTP-Username](#) function), FTPit will use the Default Username from this Basics tab.

FTPit waits for a certain amount of time for your FTP host to respond. By default, this is set to 15 seconds. However, you can adjust the setting to fit your needs by entering a different value in the **Connection Timeout** field.



The image shows a screenshot of the 'FTPit - A FileMaker Pro Plug-in' configuration dialog box. The 'Basics' tab is selected, showing default settings for FTP Host, Username, Password, and Connection Timeout (in secs). The 'Register' button is disabled, while 'Cancel' and 'OK' are active.

Field	Value
FTP Host	ftp.yourdomain.com
Username	anonymous
Password	you@yourdomain.com
Connection Timeout (in secs)	15

Figure 3. Basics tab

Advanced Tab

The Advanced Tab holds settings that you most likely will never need to change. You should not change any of the settings on this tab unless you know exactly what you are doing, or if our tech support team instructs you to change the settings.

About Tab

The **About** Tab (See Figure 2) simply tells you which version of FTPit you are currently running. It also displays who this copy of FTPit is registered to. If you do not have the most recent version that is listed on our website (<http://www.cnsplug-ins.com>), then you can upgrade simply by downloading and installing the newer version.



Figure 4. About tab

Registration

If you find FTPit useful and would like to register it, you can purchase a license to use it from our secure online purchase form. Simply go to <http://www.cnsplug-ins.com/> and click on the Purchase link. Once you have purchased a license to use FTPit, you will be sent a registration code to register your copy of the plug-in. To do that, simply open any of the example databases and press the "Register FTPit" button. and enter in your registration information. You can also register by opening the configuration dialog, and clicking the Register button.

Working with FTPit

This section of the documentation explains a few fundamentals you need to understand when using FTPit to build your own custom solutions and to understand the example solutions provided for you. First, though, if you are new to plug-ins (and maybe even if you are not), you should take a look at the "How To Use Plug-ins" document that is included with this download (and available on our website). That document explains the fundamentals of using plug-ins with FileMaker and is a must-read if you are new to plug-ins. This section and the following sections of this document expect you to understand the fundamental ideas from the "How To Use Plug-ins" document.

The single most important concept in FTPit is the idea of the "current" directory. All of the file related functions in FTPit use this concept of the "current" directory. Instead of referring to files by their full path (like "c:\My Documents\projects\my website\images\logo.gif"), you refer to them only by their filename (like "logo.gif"). This is accomplished by setting the "current" directory with the functions [doFTP-ChangeDir](#) and [doFTP-LocalChangeDir](#). [doFTP-ChangeDir](#) changes the "current" remote directory (the directory of the FTP site you are connected to), while [doFTPLocalChangeDir](#) changes the "current" local directory (the directory of your hard drive). Once you have set the "current" directory, you can use any of the file related functions on the files located in the "current" directory.

For example, let's say you wanted to upload the file "logo.gif" from the directory "c:\My Documents\projects\my website\images\" on your local hard drive to your ftp site in the directory "/users/frank/public_html/images/". The FTPit functions you would call are the following:

```
External("doFTP-LocalChangeDir", "c:\My Documents\projects\my website\images\") &  
External("doFTP-ChangeDir", "/users/frank/public_html/images/") &  
External("doFTP-Put", "logo.gif")
```

If there are other files in the same local directory that you want to upload, all you have to do is put in a few more function calls to [doFTP-Put](#), like so:

```
External("doFTP-Put", "bannerad.gif")  
External("doFTP-Put", "product1.gif")
```

Another important concept is that FTPit uploads and downloads files in the "background", meaning that while FTPit is uploading or downloading files, you can continue to work with your database. We have included a status window

in FTPit so that you know what FTPit is doing at all times. You can show, hide, and move this status window using the [doFTP-ShowStatus](#), [doFTP-HideStatus](#), and [doFTP-MoveStatus](#) functions explained in the [FTPit External Functions Reference](#) section later in this documentation.

You can also have FTPit call a script in your database when it is done uploading or downloading a file. You enable this by using the [doFTP-CompletedScript](#) function. FTPit will call the script that you define each time it has completed uploading or downloading a single file. So, if you have multiple "Puts" or "Gets", it will call the script for each one of the files you are "Putting" or "Getting". You can use the [doFTP-LastCompleted](#) function to determine which file was just completed and if you were "Putting" it or "Getting" it. You can use the [doFTP-GetQueue](#) function to see all the files that FTPit will be uploading or downloading, and to check to see if there are any more files being uploaded or downloaded. All of these functions are explained in the [FTPit External Functions Reference](#) section later in this documentation.

Version 1.5 of FTPit adds a new function named [doFTP-WaitForQueue](#). This function is used as an alternative to the [doFTP-CompletedScript](#) function, and should be used independently of that function. The [doFTP-WaitForQueue](#) function allows you to pause your script and wait for FTPit to transfer all the files in its queue before proceeding with the rest of your script. This should give you more control over your complex FTPit scripts and allow you to have a single script that does all of your processing instead of relying on a "Done" script to drive your solution.

Text Transfer Example Explained

This section of the documentation will show you, step by step, how the Text Transfer Example database works so that you can understand how to use FTPit in your own solutions. The first section describes the fields and scripts that are used in the Text Upload portion of the database. The second section describes the fields and scripts that are used in the Text Download portion of the database.

Text Upload

There are seven fields defined in the solution that are used with Text Upload portion of the database. You will not necessarily need all of these fields in your FTP solution. The fields defined are the following:

Field Name	Field Type
FTPit Result	Text
Text Body	Text
FileName	Text
Host	Text
Username	Text
Password	Text
Path	Text

The FTPit Result field is used with the Set Field script steps in the scripts. It is the field that is “set” with the results from the plug-in function calls. Host contains the domain name or IP address of the FTP site that you are connecting to. Username contains the username of the account on that FTP site. Password contains the password of the account on that FTP site. Text Body contains the “body” of the text file. You put everything in this field that you want to be in the final text file on the FTP site. Path contains the path to the directory where you want this text file to be uploaded to. Finally, File Name is the name of the file to create on the FTP site.

Upload Script

Now that you know the fields in the database, take a look at the “Upload” script. When you view this script, you see that it is calling all of the other related scripts in order, and checking for errors after each step. Look at the first related script, “Set up”:

```
Set Field [FTPit Result, External("doFTP-Host", Host) & "¶" &
  External("doFTP-Username", Username) & "¶" &
  External("doFTP-Password", Password) & "¶" &
  External("doFTP-CompletedScript", Status(CurrentFileName) & "IDone")]
```

The first three External calls set up the Host, Username, and Password using the fields in the database. The last External call tells FTPit to call the “Done” script in the current database (FTPit_Text_Upload_Example.FP5, unless you have changed the name) when it has completed uploading the text.

The next script is the “Connect” script:

```
Set Field [FTPit Result, External(“doFTP-Connect”, “”)]
```

This simply calls the [doFTP-Connect](#) function to connect to the FTP site.

The next script is the “Change Directory” script:

```
Set Field [FTPit Result, External(“doFTP-ChangeDir”, Path)]
```

This script calls the [doFTP-ChangeDir](#) function to change to the directory specified in the Path field in the database. This directory is the target directory for the file we will be uploading in the next step.

The next script is the “Put Text” script:

```
Set Field [FTPit Result, External(“doFTP-TextAssign”, Text Body)]  
Set Field [FTPit Result, External(“doFTP-PutText”, File Name)]
```

The first Set Field script step assigns the “body” of the text file using the “Text Body” field in the database. This is how you assign the contents of the file that you will be uploading/creating on the FTP site. The second Set Field script step begins the text upload to your FTP site. The parameter for the [doFTP-PutText](#) function is the name of the file that you want FTPit to create on the FTP site.

The final scripts are the “Done” and “Disconnect” scripts. The “Disconnect” script is not called from the “Upload” script, but instead is called from the “Done” script. Since FTPit transfers files in the “background”, we do not want to disconnect from the FTP server before it has finished uploading the text. This is why we set up a “CompletedScript” in the “Set up” script to call the “Done” script. When FTPit is finished uploading the text, it will call the “Done” script, which, after checking to make sure the queue is empty, calls the “Disconnect” script and tells the user it has completed uploading the text. The “Done” script looks like this:

```
If [IsEmpty(External(“doFTP-GetQueue”, “”))]  
    Perform Script [Sub-scripts, “ - Disconnect”]  
    Show Message [“The Text has been uploaded.”]  
End If
```

This first calls the [doFTP-GetQueue](#) function to see if it is empty. If the queue is empty, it then calls the “Disconnect” script to disconnect from the FTP site and then displays a message saying that the transfer is complete.

The “Disconnect” script looks like this:

```
Set Field [FTPit Result, FTPit Result & “¶” & External(“doFTP-Disconnect”, “”)]
```

This simply calls the [doFTP-Disconnect](#) function to disconnect from the FTP site. Note: The previous contents of the result field are copied back into itself when calling the [doFTP-Disconnect](#) function to allow you to see the results from the previous function call and the [doFTP-Disconnect](#) function call. This allows you to see any errors that may have caused the “Upload” script to exit before actually uploading the text (like a bad directory name).

Instead of using the “Done” script and the [doFTP-CompletedScript](#) function, you can use the new `WaitForQueue` function to have the “Upload” script pause and wait for FTPit to complete before disconnecting. The Text Download examples below use this method.

Text Download

The Text Transfer Example database also shows how you can download a text file from an FTP site directly into one or more fields in your database. You can either download the entire text file into the field at once, or you can download the text file one line at a time. This example database shows both of these methods. The first method is downloading the entire file at once. This example uses all the same fields as the Text Upload example, except that instead of using the “Text Body” field, it uses this field:

Field Name	Field Type
File Text	Text

The script that downloads the text is called “Get Whole File”. The first step in the script sets up FTPit and connects to the server. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, “¶” &
  External(“doFTP-Host”, Host) & “¶” &
  External(“doFTP-Username”, Username) & “¶” &
  External(“doFTP-Password”, Password) & “¶” &
  External(“doFTP-Connect”, “”)]
```

The first three External calls set up the Host, Username, and Password using the fields in the database. The last External call tells FTPit to Connect to the server. After this Set Field, the script checks to see if there were any errors, and if there were, it displays an error message.

If there were no errors in the first step, we move on to the second step, which changes the directory on the remote FTP site to the directory we want to download the text file from. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External(“doFTP-ChangeDir”, Path)]
```

This simply changes the remote directory to the directory specified in the Path field in the database. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site and displays an error message.

If there were no errors in the second step, we move on to the third step, which begins the download. The Set Field script that does this looks like this:

```
Set Field [FTPit Result, External(“doFTP-GetText”, File Name)]
```

This step uses the [doFTP-GetText](#) function to queue the text for download. The parameter for the [doFTP-GetText](#) function is the name of the file that you want to download. In this case, we take the name of the file from a field in

our database. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site and displays an error message.

If there were no errors in the third step, we move on to the fourth step, which simply waits for FTPit to finish downloading the text. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External("doFTP-WaitForQueue", "Dialog=Yes")]
```

This step uses the new [doFTP-WaitForQueue](#) function. This function not only pauses your script until all the files have been transferred, but also allows you to view a special version of the FTPit Status Window that shows the progress of the transfer. To get it to show that Status Window, we use the parameter "Dialog=Yes".

After FTPit downloads the text file, we need to pull the text data into the field in our database. The fifth step of our script does this with this Set Field script step:

```
Set Field [File Text, External("doFTP-TextAcquire", "")]
```

This step uses the [doFTP-TextAcquire](#) function to pull in the entire text file, and then sets the "File Text" field in our database with the text from the file.

The sixth and final step in the script simply disconnects from the FTP site. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External("doFTP-Disconnect", "")]
```

This step uses the [doFTP-Disconnect](#) function to disconnect from the FTP site.

The second text download example in the Text Transfer Example database shows how you can download a text file from the FTP Site and then extract the text one line at a time into your database. This example uses these additional fields:

Field Name	Field Type
File Line 1	Text
File Line 2	Text
File Line 3	Text

The "Get 1 Line at a Time" script is the script for this example. It is almost identical to the "Get Whole File" script. The only difference is in the fifth step. The script steps that make up the fifth step in this script are the following:

```
Set Field [File Line 1, External("doFTP-TextAcquire", "Line=1")]
Set Field [File Line 2, External("doFTP-TextAcquire", "Line=2")]
Set Field [File Line 3, External("doFTP-TextAcquire", "Line=3")]
```

Each of these Set Field script steps pull in a different line of text from the file. It does this by passing the "Line" parameter to the [doFTP-TextAcquire](#) function. You can specify any line number you need when downloading the text. If the line number you specify is greater than the number of lines in the text file, the [doFTP-TextAcquire](#) function will return "***EOF**".

Single File Transfer Example Explained

This section of the documentation will show you, step by step, how the Single File Transfer Example database works so that you can see how to use FTPit in your own solutions. This example includes both a Single File Upload Example and a Single File Download example. The fields in this database are the following:

Field Name	Field Type
FTPit Result	Text
Host	Text
Username	Text
Password	Text
Remote Path	Text
Local Path	Text
File Name	Text

The FTPit Result field is used with the Set Field script steps in the scripts. It is the field that is “set” with the results from the plug-in function calls. Host contains the domain name or IP address of the FTP site that you are connecting to. Username and Password contain the username and password of the account on that FTP site. Remote Path is the full path to the directory on the FTP site that contains or will contain the file. Local Path is the full path to the directory on your machine that contains or will contain the file. File Name is the name of the file that you will be transferring.

Upload Script

Now that you know the fields in the database, take a look at the “Upload” script. When you view this script, you see that it is calling all of the other related scripts in order, and checking for errors after each step. Look at the first related script, “Set Up”:

```
Set Field [FTPit Result, External("doFTP-Host", Host) & "¶" &  
External("doFTP-Username", Username) & "¶" &  
External("doFTP-Password", Password) & "¶" &  
External("doFTP-CompletedScript", Status(CurrentFileName) & "IDone")]
```

The first three external calls set up the Host, Username, and Password using the fields in the database. The last External call tells FTPit to call the “Done” script in the current database (FTPit_Single_Transfer_Ex.fp5, unless you have changed the name) when it has completed uploading the file.

The next script is the “Connect” script:

```
Set Field [FTPit Result, External("doFTP-Connect", "")]
```

This simply calls the [doFTP-Connect](#) function to connect to the FTP site.

The next script is the “Change Directories” script:

```
Set Field [FTPit Result, External(“doFTP-ChangeDir”, Remote Path) & “¶” &
  External(“doFTP-LocalChangeDir”, Local Path)]
```

This script calls the [doFTP-ChangeDir](#) function to change the remote directory on the FTP site to the directory specified in the Remote Path field in the database. This directory is the target directory for the file we are uploading. This script also calls the [doFTP-LocalChangeDir](#) function to change the local directory on your computer to the directory specified in the Local Path field in the database. This directory is the source directory that contains the file we are uploading.

The next script is the “Put File” script:

```
Set Field [FTPit Result, External(“doFTP-Put”, File Name)]
```

This uses the [doFTP-Put](#) function to queue the file for uploading from your computer to the FTP site. The parameter for the [doFTP-Put](#) function is the name of the file that you are uploading.

The final scripts are the “Done” and “Disconnect” scripts. The “Disconnect” script is not called from the “Upload” script, but instead is called from the “Done” script. Since FTPit transfers files in the “background”, we do not want to disconnect from the FTP server before it has finished uploading the file. This is why we set up a “CompletedScript” in the “Set up” script to call the “Done” script. When FTPit is finished uploading the file, it will call the “Done” script, which, after checking to make sure the queue is empty, calls the “Disconnect” script and tells the user it has completed uploading the file. The “Done” script looks like this:

```
If [IsEmpty(External(“doFTP-GetQueue”, “”))]
  Perform Script [Sub-scripts, “ - Disconnect”]
  Show Message [“The File has been transferred.”]
End If
```

This first calls the [doFTP-GetQueue](#) function to see if it is empty. If the queue is empty, it then calls the “Disconnect” script to disconnect from the FTP site and then displays a message saying that the transfer is complete.

The “Disconnect” script looks like this:

```
Set Field [FTPit Result, FTPit Result & “¶” &
  External(“doFTP-Disconnect”, “”)]
```

This simply calls the [doFTP-Disconnect](#) function to disconnect from the FTP site. Note: The previous contents of the result field are copied back into itself when calling the [doFTP-Disconnect](#) function to allow you to see the results from the previous function call and the [doFTP-Disconnect](#) function call. This allows you to see any errors that may have caused the “Upload” script to exit before actually uploading the file (like a bad directory name).

Instead of using the “Done” script and the [doFTP-CompletedScript](#) function, you can use the new [doFTP-WaitForQueue](#) function to have the “Upload” script pause and wait for FTPit to complete before disconnecting. You can look at the Multi File Transfer Example database for examples of how this is done.

Download Script

The “Download” script is very similar to the “Upload” script. In fact it is almost identical except that instead of calling the “Put File” script to upload a file, it calls the “Get File” script to download a file. The “Get Field” script looks like this:

```
Set Field [FTPit Result, External(“doFTP-Get”, File Name)]
```

This uses the [doFTP-Get](#) function to queue the file for downloading from the FTP site to your computer. The parameter for the [doFTP-Get](#) function is the name of the file that you are downloading.

All of the other scripts are the exact same scripts that are called by the “Upload” script and are described above.

Multiple File Transfer Example Explained

This example database shows you how to transfer all files from a given directory at once. You can also optionally provide a wildcard to only transfer certain files from the directory. (For more information on specifying a wildcard, see the [doFTP-LocalListFirst](#) function.) The fields in this database are the following:

Field Name	Field Type	Field Options
FTPit Result	Text	
Host	Text	
Username	Text	
Password	Text	
Remote Path	Text	
Local Path	Text	
Wildcard	Text	
List Result	Text	
File Info	Text	
File Name	Calculation	Middle(File Info, Position(File Info, “File Name=”, 1, 1) + (Length(“FileName=”)), Position(File Info, “¶”, Position(File Info, “FileName=”, 1, 1), 1) - (Position(File Info, “FileName=”, 1, 1) + (Length(“FileName=”))))
Dir Flag	Calculation	Middle(File Info, Position(File Info, “Dir-Flag=”, 1, 1) + (Length(“DirFlag=”)), Position(File Info, “¶”, Position(File Info, “DirFlag=”, 1, 1), 1) - (Position(File Info, “DirFlag=”, 1, 1) + (Length(“DirFlag=”))))
File Count	Number	

The FTPit Result field is used with the Set Field script steps in the scripts. It is the field that is “set” with the results from the plug-in function calls. Host contains the domain name or IP address of the FTP site that you are connecting to. Username contains the username of the account on that FTP site. Password contains the password of the account on that FTP site. Local Path contains the full path to the directory on the local computer. Remote path

contains the full path to the directory on the remote FTP site. Wildcard holds the wildcard to use when retrieving the list of files to transfer. List Result is used with the Set Field script steps that list the files to transfer. File Info contains the information for each individual file. The File Name and Dir Flag fields are calculation fields that pull the File Name and Dir Flag attributes of each file out of the File Info field. Finally, File Count contains the number of files transferred.

Multi Upload Script

This script is used to upload all the files from a directory on your computer to a directory on an FTP site. The first step is to set up the plug-in and make the initial connection to the server. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, “¶” &
          External(“doFTP-Host”, Host) & “¶” &
          External(“doFTP-Username”, Username) & “¶” &
          External(“doFTP-Password”, Password) & “¶” &
          External(“doFTP-Connect”, “”)]
```

The first three External calls set up the Host, Username, and Password using the fields in the database. The last External call tells FTPit to Connect to the server. After this Set Field, the script checks to see if there were any errors, and if there were, it displays an error message and exits the script.

The second step of the script is to change the remote directory on the FTP server. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External(“doFTP-ChangeDir”, Remote Path)]
```

This simply changes the remote directory to the directory specified in the Remote Path field in the database. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site, displays an error message, and exits the script.

The third step of the script is to change the local directory on the local computer. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External(“doFTP-LocalChangeDir”, Local Path)]
```

This simply changes the local directory to the directory specified in the Local Path field in the database. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site, displays an error message, and exits the script.

The fourth step of the script is to try and get a list of files to transfer. The Set Field script step that does this looks like this:

```
Set Field [List Result, External(“doFTP-LocalListFirst”, Wildcard)]
```

This uses the [doFTP-LocalListFirst](#) function (with the optional wildcard from the field) to grab a list of files from your

computer. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site, displays an error message, and exits the script.

The fifth step of the script is to iterate through all of the files in the local directory and queue them for uploading to the FTP site. The script steps in this section of the script look like the following:

```
Set Field [File Count, 0]
Loop
  Set Field [File Info, List Result]
  If [Dir Flag <> "d"]
    #We can only upload files, not directories.
    Set Field [FTPit Result, External("doFTP-Put", File Name)]
    Set Field [File Count, File Count + 1]
  End If
  Set Field [List Result, External("doFTP-LocalListNext", "")]
  Exit Loop If [IsEmpty(List Result)]
End Loop
```

The first Set Field in this section initializes the File Count to zero so that we can count the number of files that we transfer. The script then goes into a loop. Inside the loop, the first Set Field takes the results from the [doFTP-LocalListFirst](#) function (that we previously put into the List Result field) and puts them into the File Info field. This allows the File Name and Dir Flag calculation fields to extract their proper values about the first file. The script then checks the Dir Flag field to make sure that it is not a "d". If Dir Flag contains the value "d", this means that the first file was actually a directory, which we cannot upload. If the Dir Flag field contains anything else, then we know it is a file, so the next Set Field script step uses the [doFTP-Put](#) function to queue the file for transferring. The next Set Field increments our File Count field. After the End If, the next Set Field uses the [doFTP-LocalListNext](#) function to retrieve the File Info for the next file in the directory. If the [doFTP-LocalListNext](#) function returns nothing, then we know there are no more files. So, we use the IsEmpty() function in an Exit Loop If script step to exit out of the loop if the [doFTP-LocalListNext](#) function returns nothing. Otherwise, we loop back up to the beginning to queue the next file. After this section of the script, the script checks the File Count field to see if it still equals zero. If it does, then we know that the plug-in could not find any files in the local directory, so it disconnects from the server, displays an error message, and exits the script.

The sixth step of the script waits for FTPit to transfer all the files we have just queued. This uses the new function [doFTP-WaitForQueue](#). This function not only pauses your script until all the files have been transferred, but also allows you to view a special version of the FTPit Status Window that shows the progress of the transfer. Instead of doing it this way, you could also use the [doFTP-CompletedScript](#) function and a "Done" script to allow FTPit to not take over your copy of FileMaker, but instead quietly transfer the files in the background and inform you when it's done by calling a script in your database. For an example of this method, look at the Single File Transfer Example database.

The seventh and final step of the script disconnects from the server. The Set Field script step that does this looks like this:

```
Set Field [FTPit Result, External("doFTP-Disconnect", "")]
```

This simply uses the [doFTP-Disconnect](#) function to disconnect from the FTP site since we have transferred all the

files and are done with the connection. After this Set Field, the script displays a message saying the transfer was complete, and that is the end of the script.

Multi Download Script

This script is almost identical to the Multi Upload Script. The only difference is that this file grabs a list of the remote files and queues the files for download instead of upload. These are the effected steps:

The forth step of the script is to try and get a list of files to transfer. The Set Field script step that does this looks like this:

```
Set Field [List Result, External("doFTP-ListFirst", Wildcard)]
```

This uses the [doFTP-ListFirst](#) function (with the optional wildcard from the field) to grab a list of files from the remote FTP site. After this Set Field, the script checks to see if there was an error, and if there was, it disconnects from the FTP site, displays an error message, and exits the script.

The fifth step of the script is to iterate through all of the files in the remote directory and queue them for downloading to your computer. The script steps in this section of the script look like the following:

```
Set Field [File Count, 0]
Loop
  Set Field [File Info, List Result]
  If [Dir Flag <> "d"]
    #We can only download files, not directories.
    Set Field [FTPit Result, External("doFTP-Get", File Name)]
    Set Field [File Count, File Count + 1]
  End If
  Set Field [List Result, External("doFTP-ListNext", "")]
  Exit Loop If [IsEmpty(List Result)]
End Loop
```

The first Set Field in this section initializes the File Count to zero so that we can count the number of files that we transfer. The script then goes into a loop. Inside the loop, the first Set Field takes the results from the [doFTP-ListFirst](#) function (that we previously put into the List Result field) and puts them into the File Info field. This allows the File Name and Dir Flag calculation fields to extract their proper values about the first file. The script then checks the Dir Flag field to make sure that it is not a "d". If Dir Flag contains the value "d", this means that the first file was actually a directory, which we cannot download. If the Dir Flag field contains anything else, then we know it is a file, so the next Set Field script step uses the [doFTP-Get](#) function to queue the file for transferring. The next Set Field increments our File Count field. After the End If, the next Set Field uses the [doFTP-ListNext](#) function to retrieve the File Info for the next file in the directory. If the [doFTP-ListNext](#) function returns nothing, then we know there are no more files. So, we use the IsEmpty() function in an Exit Loop If script step to exit out of the loop if the [doFTP-ListNext](#) function returns nothing. Otherwise, we loop back up to the beginning to queue the next file. After this section of the script, the script checks the File Count field to see if it still equals zero. If it does, then we know that the plug-in could not find any files in the remote directory, so it disconnects from the server, displays an error message, and exits the script.

All of the other steps in this script are identical to the Multi Upload script, which you can find explanations for above.

FTPit External Function Reference

Available Functions

The following is a list of all available functions and a complete description of each. For general information on how the External () functions work and how you can use them in your calculations, please see the How to use Plug-ins PDF Document distributed with this Document.

Functions with an asterisk (*) are new with version 1.5. Functions with a dagger (†) have changed with version 1.5.

doFTP-Append *

This function works like the [doFTP-Put](#) function except instead of uploading a new file, or overwriting an exiting file, it appends the file data to an existing file on the server. The file name you specify should already exist on the server.

Example:

```
External("doFTP-Append", "FileName")
```

See Also:

[doFTP-Get](#)
[doFTP-GetText](#)
[doFTP-Put](#)
[doFTP-PutText](#)

doFTP-ChangeDir

This function changes the "current" remote directory to the path that you specify. This is the only function that takes a full path to a directory on the FTP site. All of the other functions work with files in the "current" remote directory that you set with this function. See the Working with FTPit section of the FTPit documentation for a more complete understanding.

Examples:

```
External("doFTP-ChangeDir", "/pub/downloads/")  
External("doFTP-ChangeDir", "/users/frank/public_html/")
```

See Also:

[doFTP-ChangeDirUp](#)
[doFTP-CurrentDir](#)
[doFTP-MakeDir](#)
[doFTP-RemoveDir](#)

doFTP-ChangeDirUp

This function is a short cut to changing the current remote directory to its parent directory. For instance, if the current remote directory is `"/pub/downloads/"`, after calling this function, the current remote directory would be `"/pub/"`. The parameter should be the empty string (`""`).

Examples:

```
External("doFTP-ChangeDirUp", "")
```

See Also:

[doFTP-ChangeDir](#)
[doFTP-CurrentDir](#)
[doFTP-MakeDir](#)
[doFTP-RemoveDir](#)

doFTP-CompletedScript t

Use this function to tell FTPit which script you would like it to call when it has completed uploading or downloading a file or if there was an error uploading or downloading a file. This function should be used independently and separately from the [doFTP-WaitForQueue](#) function. If `doFTP-CompletedScript` is confusing, look at the [doFTP-WaitForQueue](#) function to see if it makes more sense. The parameter consists of the database the script is in, as well as the name of the script, separated by the pipe character (`"|"`). (`"|"` is created by typing shift-backslash.) You must specify the full name of the database, including the `".fp3"` or `".fp5"` extension. (Macintosh users may not have the extension, depending on how you originally named the file.)

Examples:

```
External("doFTP-CompletedScript", "FTPit_Text_Upload_Example.fp3|Done")
```

See Also:

[doFTP-LastCompleted](#)
[doFTP-WaitForQueue](#)

doFTP-Connect

This function opens a connection to the FTP site you defined as the Host and uses the Username and Password to log in. The parameter for this function should be the empty string ("").

Examples:

```
External("doFTP-Connect", "")
```

See Also:

[doFTP-Host](#)

[doFTP-Username](#)

[doFTP-Password](#)

[doFTP-Disconnect](#)

doFTP-CurrentDir

This function returns the current remote directory. This function is useful if the FTP site you log into automatically redirects you to your home directory. It is also useful if you are connected to a unix FTP site and you change directory to a "link". In general, it is a good idea to call doFTP-CurrentDir after calling [doFTP-ChangeDir](#) or [doFTP-ChangeDirUp](#) if you need to know exactly where you are on the FTP site at all times. The parameter should be the empty string ("").

Examples:

```
External("doFTP-CurrentDir", "")
```

See Also:

[doFTP-ChangeDir](#)

[doFTP-ChangeDirUp](#)

[doFTP-MakeDir](#)

[doFTP-RemoveDir](#)

doFTP-DataPortRangeEnd *

This function sets the End of the Port Range for the Data Connection. This is an advanced function for when you must connect to the internet through a firewall. You will most likely never need to use this function.

Example:

```
External("doFTP-DataPortRangeEnd", "59999")
```

See also:

[doFTP-DataPortRangeStart](#)

doFTP-DataPortRangeStart *

This function sets the Start of the Port Range for the Data Connection. This is an advanced function for when you must connect to the internet through a firewall. You will most likely never need to use this function.

Example:

```
External("doFTP-DataPortRangeStart", "59000")
```

See also:

[doFTP-DataPortRangeEnd](#)

doFTP-Delete

Use this function to delete a file in the current remote directory. Only specify the filename and not a full path.

Examples:

```
External("doFTP-Delete", "index_backup.htm")
```

doFTP-Disconnect

This function closes the connection to the FTP site. The parameter for this function should be the empty string ("").

Examples:

```
External("doFTP-Disconnect", "")
```

See Also:

[doFTP-Connect](#)

doFTP-Get

This function retrieves a file from the current remote directory and saves it to the current local directory. You only specify the filename in the parameter, and not a full path. If the file is not in the current remote directory, use [doFTP-ChangeDir](#) first. Use [doFTP-LocalChangeDir](#) to specify where to save the file on the local hard drive.

If you call doFTP-Get with a new file while FTPit is uploading or downloading a separate file in the background, the new file will be added to a "queue" of files to be downloaded. When the current file has completed uploading or

downloading, the new file will begin to download. You can use the [doFTP-GetQueue](#) function to return the current files that will be uploaded or downloaded.

Examples:

```
External("doFTP-Get", "installer.zip")
```

See Also:

[doFTP-Type](#)
[doFTP-Passive](#)
[doFTP-Put](#)
[doFTP-PutText](#)
[doFTP-GetQueue](#)
[doFTP-ChangeDir](#)
[doFTP-LocalChangeDir](#)
[doFTP-GetText](#)
[doFTP-Append](#)

doFTP-GetLocalRoot

This function returns the "root" of the local hard drive. On windows, this function will always return "C:\". On macintosh, this function will return the name of the first "fixed" disk (your hard drive). The parameter should be the empty string ("").

Example:

```
External("doFTP-GetLocalRoot", "")
```

See Also:

[doFTP-GetPathToDB](#)
[doFTP-GetPathToFM](#)

doFTP-GetPathToDB

This function returns the path to a specified database that is open and on the local drive. You must specify the full name of the database, including the ".fp3" or ".fp5" extension.

Example:

```
External("doFTP-GetPathToDB", "FTPit_Text_Upload_Example.fp5")
```

See Also:

[doFTP-GetLocalRoot](#)
[doFTP-GetPathToFM](#)

doFTP-GetPathToFM

This function returns the path of the directory to the current version of FileMaker. The parameter should be the empty string ("").

Example:

```
External("doFTP-GetPathToFM", "")
```

See Also:

[doFTP-GetLocalRoot](#)
[doFTP-GetPathToDB](#)

doFTP-GetQueue

This function returns a list of the files you have queued for upload or download. You can queue multiple files for upload or download with multiple calls to [doFTP-Get](#), [doFTP-Put](#), and [doFTP-PutText](#). The parameter should be the empty string ("").

You can also use this function to determine if FTPit is uploading or downloading anything. This function will return the empty string ("") when there is nothing on the queue, meaning nothing is uploading or downloading. It would be a good idea to check if there is anything uploading or downloading before you disconnect. If you disconnect while a file is being uploaded or downloaded, the file will only be partially uploaded or downloaded. Here is an example:

```
Set Field ["Queue", "External("doFTP-GetQueue", "")"]
If ["Queue <> """]
    Show Message ["You are currently uploading or downloading a file. Are you sure you want to disconnect?"]
    If ["Status(CurrentMessageChoice) = 1"]
        Exit Script
    End If
End If
Set Field ["FTPit Result", "External("doFTP-Disconnect", "")"]
```

Example:

```
External("doFTP-GetQueue", "")
```

See Also:

[doFTP-Get](#)
[doFTP-Put](#)

[doFTP-PutText](#)
[doFTP-WaitForQueue](#)

doFTP-GetText *

This function works like the [doFTP-Get](#) function, except instead of downloading the file to the hard drive, it downloads the file to the internal text block in the plug-in. You can then use the [doFTP-TextAcquire](#) to read that data into fields in your database. The purpose of this function is to download text files from the FTP server and pull that text directly into a field in your database. You should only use this function to download text files. If you attempt to download a file that has binary data in it instead of plain text data, the results are undefined. Most likely you will not get anything at all.

Example:

```
External("doFTP-GetText", "FileName")
```

See also:

[doFTP-Append](#)
[doFTP-Get](#)
[doFTP-Put](#)
[doFTP-PutText](#)
[doFTP-TextAcquire](#)

doFTP-HideStatus

Use this function to hide the status window that you had previously shown with the [doFTP-ShowStatus](#) function. You must call this function to hide the status window before closing the FileMaker Pro application (see [doFTP-ShowStatus](#) above). Power-users will see that this function returns the last known coordinates of the status window, which can be extracted and stored so that the next time the status window is shown, it can be shown in the same place. The parameter should be the empty string ("").

Example:

```
External("doFTP-HideStatus", "")
```

See Also:

[doFTP-ShowStatus](#)
[doFTP-MoveStatus](#)

doFTP-Host

The Host is the domain name or IP address of the FTP site you are connecting to. You can assign the host in your scripts, or set up a Default Host in the Configuration Dialog. [doFTP-Connect](#) will return an error if no host has been set. If you assign a host with this function, FTPit will ignore the Default Host in the Configuration Dialog.

Example:

```
External("doFTP-Host", "ftp.filemaker.com")
```

See Also:

[doFTP-Username](#)

[doFTP-Password](#)

[doFTP-Connect](#)

doFTP-IsConnected *

Use this function to determine whether you are connected to your FTP site. The function will return "YES" if you are connected, and "NO" if you are not connected. The parameter for this function should be the empty string ("").

Example:

```
External("doFTP-IsConnected", "")
```

doFTP-LastCompleted

The doFTP-LastCompleted function tells you what file FTPit just completed uploading or downloading or the error that occurred when it tried to upload or download. You would use this function inside the script that the [doFTP-CompletedScript](#) calls. The parameter for this function should be the empty string (""). This function returns text in these forms:

GET: downloadedfilename.txt

PUT: uploadedfilename.txt

ERROR: GET: <error code>

ERROR: PUT: <error code>

The first two are returned telling you that it successfully uploaded or downloaded a file. The second two are returned if there was an error uploading or downloading the file.

Example:

```
External("doFTP-LastCompleted", "")
```

See Also:

[doFTP-CompletedScript](#)

[doFTP-WaitForQueue](#)

doFTP-ListFirst

Use this function in conjunction with `doFTP-ListNext` to list all of the files in the current remote directory. `doFTP-ListFirst` will pull the complete file list from the FTP site, parse it into a useful list of files, and return the first file in the list. If there are no files to list, this function will return "ERROR: ListFirst: No Files." You should not call [doFTP-ListNext](#) if this function returns an error (doing so will do no harm, it just will not return any useful information). Refer to the `FTPit_Example` and `FTPit_Files` databases for an example on how to use `doFTP-ListFirst` and [doFTP-ListNext](#).

Under normal circumstances, the parameter is an empty string (""). However, you can also specify a "wild card" to match against the filenames so that you only return a subset of files. For instance, if you only want to see the HTML files in the directory, you could specify a wild card like "*.htm". The wild cards are based on the wild cards you can use with the MSDos "Dir" command, if you are familiar with those. Here is a brief description:

- "*" - Matches zero or more characters
- "?" - Matches only one character
- Any other character matches itself.

If you wanted to display all FileMaker Pro 4 and 5 files, you could specify the wild card "*.fp?". The "*" would match any filename, while the "fp?" would match "fp3" and "fp5". You should also note, that you may not always get what you want. For instances, the "*.fp?" wildcard could also match a file with the name "obscure.fpk".

Examples:

```
External("doFTP-ListFirst", "")
External("doFTP-ListFirst", "*.fp?")
```

See Also:

[doFTP-ListNext](#)

doFTP-ListNext

Use this function to get the remaining filenames in the current remote directory after calling [doFTP-ListFirst](#). When this function returns the empty string (""), there are no more filenames to retrieve. The parameter should be the empty string ("").

Example:

```
External("doFTP-ListNext", "")
```

See Also:

[doFTP-ListFirst](#)

doFTP-LocalChangeDir

This function changes the "current" local directory to the path that you specify. This is the only function that takes a full path to a directory on your hard drive. All of the other functions work with files in the "current" local directory that you set with this function.

Examples:

```
External("doFTP-LocalChangeDir", "c:\myfiles\  
External("doFTP-LocalChangeDir", "Macintosh HD:myfiles:")
```

See Also:

[doFTP-LocalChangeDir](#)
[doFTP-LocalCurrentDir](#)
[doFTP-LocalMakeDir](#)
[doFTP-LocalRemoveDir](#)

doFTP-LocalChangeDirUp

This function is a short cut to changing the current local directory to its parent directory. For instance, if the current local directory is "c:\myfiles\downloads\" or "Macintosh HD:myfiles:downloads:", after calling this function, the current local directory would be "c:\myfiles\" or "Macintosh HD:myfiles:". The parameter should be the empty string ("").

Example:

```
External("doFTP-LocalChangeDirUp", "")
```

See Also:

[doFTP-LocalChangeDir](#)
[doFTP-LocalCurrentDir](#)
[doFTP-LocalMakeDir](#)
[doFTP-LocalRemoveDir](#)

doFTP-LocalCurrentDir

This function returns the current local directory. The parameter should be the empty string ("").

Example:

```
External("doFTP-LocalCurrentDir", "")
```

See Also:

[doFTP-LocalChangeDir](#)

[doFTP-LocalChangeDirUp](#)
[doFTP-LocalMakeDir](#)
[doFTP-LocalRemoveDir](#)

doFTP-LocalDelete

Use this function to delete a file in the current local directory. Only specify the filename and not a full path.

Example:

```
External("doFTP-LocalDelete", "index_backup.htm")
```

doFTP-LocalListFirst

Use this function in conjunction with [doFTP-LocalListNext](#) to list all of the files in the current local directory. doFTP-LocalListFirst will pull the complete file list from your hard drive, parse it into a useful list of files, and return the first file in the list. If there are no files to list, this function will return "ERROR: LocalListFirst: No Files." You should not call [doFTP-LocalListNext](#) if this function returns an error (doing so will do no harm, it just will not return any useful information). Refer to the FTPit_Example and FTPit_Files databases for an example on how to use doFTP-LocalListFirst and [doFTP-LocalListNext](#).

Under normal circumstances, the parameter is an empty string (""). However, you can also specify a "wild card" to match against the filenames so that you only return a subset of files. For an example of wild cards, see the [doFTP-LocalListFirst](#) function above.

Examples:

```
External("doFTP-LocalListFirst", "")  
External("doFTP-LocalListFirst", "*.fp?")
```

See Also:

[doFTP-LocalListNext](#)

doFTP-LocalListNext

Use this function to get the remaining files in the current local directory after calling [doFTP-LocalListFirst](#). When this function returns the empty string (""), there are no more files to retrieve. The parameter should be the empty string ("").

Example:

```
External("doFTP-LocalListNext", "")
```

See Also:

[doFTP-LocalListFirst](#)

doFTP-LocalMakeDir

Use this function to create a new directory in the current local directory. Only specify the name of the new directory and not a full path. If you are creating a directory to download files to, remember to call [doFTP-LocalChangeDir](#) to change the current local directory to the new directory before calling [doFTP-Get](#).

Example:

```
External("doFTP-LocalMakeDir", "new_dir")
```

See Also:

[doFTP-LocalChangeDir](#)

[doFTP-LocalChangeDirUp](#)

[doFTP-LocalCurrentDir](#)

[doFTP-LocalRemoveDir](#)

doFTP-LocalRemoveDir

Use this function to delete a directory in the current local directory. Only specify the name of the directory and not a full path. You cannot delete a directory if there are files in the directory; you would need to delete all the files in the directory before removing it.

Example:

```
External("doFTP-LocalRemoveDir", "old_backup")
```

See Also:

[doFTP-LocalChangeDir](#)

[doFTP-LocalChangeDirUp](#)

[doFTP-LocalCurrentDir](#)

[doFTP-LocalMakeDir](#)

doFTP-LocalRename

Use this function to rename a file in the current local directory. Only specify the filenames and not full paths. Specify the current name followed by the new name, separated with the colon (":"). For instance, if the file is currently named "index.htm" and you want to rename it to "index_backup.htm", use the parameter "index.htm:index_backup.htm". (Note: The colon character (":") is used to separate the files instead of the pipe character ("|"). The reason is because while a pipe character can be used in a filename on macintosh and windows, a colon cannot. This allows you to rename a file with a pipe character in the name.)

Example:

```
External("doFTP-LocalRename", "index.htm:index_backup.htm")
```

doFTP-LocalSize

Use this function to return the size of a file in the current local directory. Only specify the filename and not a full path.

Example:

```
External("doFTP-LocalSize", "installer.zip")
```

doFTP-MakeDir

Use this function to create a new directory in the current remote directory. Only specify the name of the new directory and not a full path. If you are creating a directory to put files in, remember to call [doFTP-ChangeDir](#) to change the current remote directory to the new directory before calling [doFTP-Put](#) or [doFTP-PutText](#). The doFTP-MakeDir function will now create multiple subdirectories if they do not exist. In other words, if you tell the doFTP-MakeDir function to create the directory "/graphics/thumbnails", then FTPit will create both the "graphics" and the "thumbnails" directories if they do not exist.

Example:

```
External("doFTP-MakeDir", "new_dir")
```

See Also:

[doFTP-ChangeDir](#)
[doFTP-ChangeDirUp](#)
[doFTP-CurrentDir](#)
[doFTP-RemoveDir](#)

doFTP-MoveStatus

Use this function to move the status window to a different location on the screen. The parameter is the same as the parameter defined in the [doFTP-ShowStatus](#) function above. Power-users will see that this function returns the coordinates before and after the move, which can be extracted and stored so that the status window can be moved back to its previous location if desired.

Examples:

```
External("doFTP-MoveStatus", "700,100")  
External("doFTP-MoveStatus", "-1,600")  
External("doFTP-MoveStatus", "-1,-1")
```

See Also:

[doFTP-ShowStatus](#)

[doFTP-HideStatus](#)

doFTP-Passive

This function sets the connection type to Passive or Non-Passive for data transfers. By default this is Off, and you will most likely never have to change it. However, some FTP sites require Passive connections, and some Proxy servers require that you set it to On. So, if you are having trouble connecting to an FTP site, or you are using a Proxy server that requires it, call this function with a parameter of "On". Giving the parameter "Off" will turn Passive Mode back off.

Examples:

```
External("doFTP-Passive", "On")
```

```
External("doFTP-Passive", "Off")
```

doFTP-Password

Use the doFTP-Password function to assign the password of the FTP account you are connecting to. If you are connecting to an anonymous FTP site, use your email address.

Examples:

```
External("doFTP-Password", "love")
```

```
External("doFTP-Password", "frank@domain.com")
```

See Also:

[doFTP-Host](#)

[doFTP-Username](#)

[doFTP-Connect](#)

doFTP-Port

Use the doFTP-Port function to assign the port that FTPit uses to connect to an FTP site. By default this is set to port 21, and you will most likely never need to use this function.

Example:

```
External("doFTP-Port", "21")
```

doFTP-Put

This function takes a file from the current local directory and uploads it to the current remote directory. You only specify the filename in the parameter, and not a full path. If the file is not in the local directory, use [doFTP-LocalChangeDir](#) first. Use [doFTP-ChangeDir](#) to specify where to upload the file on the FTP site.

If you call doFTP-Put with a new file while FTPit is uploading or downloading a separate file in the background, the new file will be added to a "queue" of files to be uploaded. When the current file has completed uploading or downloading, the new file will begin to upload. You can use the [doFTP-GetQueue](#) function to return the current files that will be uploaded or downloaded.

Example:

```
External("doFTP-Put", "index.htm")
```

See Also:

[doFTP-Type](#)
[doFTP-Passive](#)
[doFTP-Get](#)
[doFTP-PutText](#)
[doFTP-GetQueue](#)
[doFTP-ChangeDir](#)
[doFTP-LocalChangeDir](#)
[doFTP-GetText](#)
[doFTP-Append](#)

doFTP-PutText

This function uploads the "Text" that you assigned with [doFTP-TextAssign](#) and [doFTP-TextAppend](#). You specify the filename to upload it as in the parameter. Use [doFTP-ChangeDir](#) to specify where to upload the text on the FTP site.

If you call doFTP-PutText with some new text while FTPit is uploading or downloading a separate file or text in the background, the new text will be added to a "queue" of files to be uploaded. When the current file has completed uploading or downloading, the new text will begin to upload. You can use the [doFTP-GetQueue](#) function to return the current files that will be uploaded or downloaded.

Examples:

```
External("doFTP-PutText", "products.htm")
```

See Also:

[doFTP-Get](#)
[doFTP-Put](#)

[doFTP-TextAssign](#)
[doFTP-TextAppend](#)
[doFTP-GetQueue](#)
[doFTP-ChangeDir](#)
[doFTP-TextAcquire](#)
[doFTP-GetText](#)
[doFTP-Append](#)

doFTP-Quote

Use this function to send UNIX commands to the FTP server. Doing a "chmod" or any other UNIX command is not in the actual FTP specification. However, some FTP servers allow you to use the FTP 'SITE' command to send UNIX commands to the server. If you understand that, then you can use the "doFTP-Quote" function to send that SITE command to the FTP server. FTPit will send whatever you specify as a parameter directly to the FTP server you are connected to. This allows you to send any FTP commands that FTPit does not directly have functions for. Note though, that FTPit does not currently return any results from the command, so it can really only be used to "tell" the FTP server something, and not "ask" it something.

Example:

```
External("doFTP-Quote", "SITE chmod 775 myfile.txt")
```

doFTP-Register †

This function allows you to register your copy of FTPit via a function rather than using the Configuration Dialog. It is mostly meant for developers so that they can register plug-ins for bound solutions. For people who have Site Licenses that have a startup script that registers the plugin every time the solution starts, you can use the new "I Accept the License Agreement" parameter to automatically accept the new license agreement that pops up when the plugin is registered. The parameter consists of your first name, last name, serial number, and optional license agreement all separated by the pipe character ("|"). ("|" is created by typing shift-backslash.) You can also use "Dialog" as the parameter to make the register window open.

Examples:

```
External("doFTP-Register", "First Name|Last Name|Serial Number")  
External("doFTP-Register", "First Name|Last Name|Serial Number|I Accept the License Agreement")  
External("doFTP-Register", "Dialog")
```

doFTP-RemoveDir

Use this function to delete a directory in the current remote directory. Only specify the name of the directory and not a full path. In general, FTP sites will not allow you to delete a directory if there are files in the directory; you would need to delete all the files in the directory before removing it.

Example:

```
External("doFTP-MakeDir", "old_backup")
```

See Also:

[doFTP-ChangeDir](#)
[doFTP-ChangeDirUp](#)
[doFTP-CurrentDir](#)
[doFTP-MakeDir](#)

doFTP-Rename

Use this function to rename a file in the current remote directory. Only specify the filenames and not full paths. Specify the current name followed by the new name, separated with the colon (":"). For instance, if the file is currently named "index.htm" and you want to rename it to "index_backup.htm", use the parameter "index.htm:index_backup.htm". (Note: The colon character (":") is used to separate the files instead of the pipe character ("|"). The reason is because while a pipe character can be used in a filename on macintosh and windows, a colon cannot. This allows you to rename a file with a pipe character in the name.)

Example:

```
External("doFTP-Rename", "index.htm:index_backup.htm")
```

doFTP-ShowStatus

Use this function to show a status window that displays information about what the plug-in is currently doing. For instance, when you are uploading or downloading a file, the status window will show you the file it is uploading or downloading as well as a progress bar indicating how much of the file has been uploaded or downloaded. If you specify an empty string ("") as the parameter, the status window will show up in the middle of the screen.

If you want to specify a starting location for the status window, specify the coordinates of the left and top of the dialog in pixels in the form "x,y" or "across,down". For instance, if you wanted to display it in the top right hand corner of your screen, and your screen resolution is set to 800x600, you could specify "700,100". If you specify a negative one ("-1") as either the x (across) or y (down) coordinates, the status window will be centered on that axis. For instance, if you want to display it on the bottom of your screen in the center, you would specify "-1,600"; or in the center of the screen by specifying "-1,-1".

On windows, this status window will always be on top of every other window and should never be hidden by another window. On macintosh, the status window can be hidden behind another open window. You can use the "Window" menu in the menu bar across the top of your screen to bring the status window to the front.

Also, on macintosh, you must ensure that the status window is closed before quitting the FileMaker Pro application. If the status window is still visible when you close the FileMaker Pro application, you will get an error message saying that FileMaker Pro cannot write to the disk, and give you the option to "Quit" or "Continue". You will only be

able to "Quit", which is the equivalent of a force-quit, which does not safely free all its resources. The easiest way to ensure that the status window is closed when you close the FileMaker Pro application is to call [doFTP-HideStatus](#) in a "close" script that you define in Edit->Preferences->Document of your main database.

Example:

```
External("doFTP-ShowStatus", "")
External("doFTP-ShowStatus", "700,100")
External("doFTP-ShowStatus", "-1,600")
```

See Also:

[doFTP-HideStatus](#)
[doFTP-MoveStatus](#)

doFTP-Size

Use this function to return the size of a file in the current remote directory. Only specify the filename and not a full path.

Example:

```
External("doFTP-Size", "installer.zip")
```

doFTP-TextAcquire *

This function allows you to retrieve the text data from the internal text block after using the [doFTP-GetText](#) function to download a text file from the FTP server. If you specify an empty parameter (""), then the plug-in will return to you all the text data up to 64000 characters (the limit on the number of characters in a FileMaker field). If the text file you downloaded is larger than 64000 characters, or if you want to read in the text piece by piece, you have two other options. You can either use the Start and Amount parameters, or the Line parameter.

The Start and Amount parameters allow you to define the starting character position you want to begin extracting from and the number of characters you want to extract. In the example below, it would return 1000 characters from the text starting at the 2000th character position. If you request an amount of characters that is greater than the amount available, it will return the number of characters that are available. In the example below, if there were only 2500 characters in the data, then the function would return only 500 characters. If you specify a Starting character position that is greater than the number of characters in the data, then the plug-in will return "***EOF**" to indicate that the end of the file has been reached. In the example below, if the data was only 1500 characters in length, then the plug-in would return "***EOF**" when you tried to request the data starting at 2000. Finally, the Start parameter should be 1-based, meaning if you want to start at the very first character, you should use "Start=1".

The Line parameter allows you to retrieve the text data one line at a time. A line is defined as any string of characters terminated by a Carriage Return and/or Line Feed character. In FileMaker, this is also known as the Paragraph Mark ("¶"). In the example below, the plug-in would return to you the 47th line in the text data. If you specify a line number that is greater than the number of lines in the text data, the plug-in will return "***EOF**" to indicate that

the end of the file has been reached. In the example below, if the data only had 40 lines, the plug-in would return "***EOF***" when you tried to request the line at position 47. Finally, the Line parameter should be 1-based, meaning if you want the very first line, you should use "Line=1".

Examples:

```
External("doFTP-TextAcquire", "")
External("doFTP-TextAcquire", "Start=2000; Amount=1000")
External("doFTP-TextAcquire", "Line=47")
```

See also:

[doFTP-GetText](#)
[doFTP-PutText](#)
[doFTP-TextAppend](#)
[doFTP-TextAssign](#)

doFTP-TextAppend

Use this function when you are uploading text from a FileMaker Pro field to an FTP site. This function adds the parameter you gave to the current "Text" that you assigned with [doFTP-TextAssign](#). This helps you break the 64k limit of text imposed on FileMaker Pro fields if you have large amounts of text you need to upload. This can also be used in a record-looping script to add extra lines for each record. If you call doFTP-TextAppend and the current "Text" is empty, this will become the initial "Text" (in other words, you don't necessarily have to call [doFTP-TextAssign](#) the very first time).

Examples:

```
External("doFTP-TextAppend", "<li>" & ProductName & "</li>")
External("doFTP-TextAppend", AFieldInYourDatabase)
```

See Also:

[doFTP-TextAssign](#)
[doFTP-PutText](#)
[doFTP-TextAcquire](#)

doFTP-TextAssign

Use this function when you are uploading text from a FileMaker Pro field to an FTP site (like in a custom static web page uploading database). This function clears out the current "Text" and assigns the parameter you gave. If you call this function twice, you will overwrite the text from the first call.

Examples:

```
External("doFTP-TextAssign", "<HTML><Body>This is the first part")
External("doFTP-TextAssign", AFieldInYourDatabase)
```

See Also:

[doFTP-TextAppend](#)
[doFTP-PutText](#)
[doFTP-TextAquire](#)

doFTP-Type

This function sets the transfer type of the connection. The options are "BINARY" and "ASCII". Binary transfer type is the default and should always be used if you are downloading any sort of application or data file (like a database). You would use the ASCII transfer type when you are uploading or downloading a text file and you want the line endings to be converted to or from your local platform. Macintosh users have an extra option, "MACBINARY", which, if the FTP site supports it, will transfer both the data fork and resource fork of the file. (If you choose "MACBINARY" in the windows version, it will default back to "BINARY".)

Examples:

```
External("doFTP-Type", "ASCII")
External("doFTP-Type", "BINARY")
External("doFTP-Type", "MACBINARY") (Macintosh only)
```

doFTP-Username

Use the doFTP-Username function to assign the username of the FTP account you are connecting to. If you are connecting to an anonymous FTP site, use "anonymous".

Examples:

```
External("doFTP-Username", "frank")
External("doFTP-Username", "anonymous")
```

See Also:

[doFTP-Host](#)
[doFTP-Password](#)
[doFTP-Connect](#)

doFTP-Version

This function returns the current version of FTPit when the parameter string is empty (""). You can also use this function to display the configuration dialog. If you pass it the parameter "CONFIGURE", the configuration dialog will open. If you pass it the parameter "ABOUT", it will display the configuration dialog with the "About" tab as the first tab shown.

Examples:

```
External("doFTP-Version", "")  
External("doFTP-Version", "CONFIGURE")  
External("doFTP-Version", "ABOUT")
```

doFTP-WaitForQueue *

This function provides an alternative to the [doFTP-CompletedScript](#) and [doFTP-LastCompleted](#) functions that have confused some of our users in past versions. The part that has confused some of our users is that FTPit does all of its uploading and downloading of files in the “background”. This allows FileMaker and your computer to be more responsive while FTPit is working. However, that method also requires that you set up a “Completed Script” for FTPit to call when it has completed a transfer so that you know that it is done and can either set up other files for transfer or disconnect from the server. This new doFTP-WaitForQueue function allows your scripts to pause and wait for FTPit to complete one or all files in the queue before moving on. This new function should be used independently and separately from the [doFTP-CompletedScript](#) function.

If you specify the empty parameter ("") for this function, then FTPit will pause your script until every file in the queue is completely finished. (If there are no files in the queue, this function will return “Queue is empty.”) So, if you queue five files for downloading and then call doFTP-WaitForQueue, then FTPit will download all 5 files, and then return to your script with the response “Queue is now empty.” You could then safely call the [doFTP-Disconnect](#) function because you will know that all the files are downloaded. You should note, though, that if you have a lot of files in the queue, or if the files are large, then FileMaker (and possibly your whole computer) will appear to be unresponsive. (This is why the original version of FTPit included the [doFTP-CompletedScript function](#) and the whole reason FTPit transfers files in the “background”.) However, this function has a few more parameters that you can use to make FileMaker seem more responsive and show that it is actually doing something.

The first parameter is the “WaitFor” parameter. This looks like either “WaitFor=All” or “WaitFor=One”. The “All” value is the same as described above when you use the empty parameter (""). It means that FTPit will pause your script until all files in the queue are completed. If you specify “One” as the value, then FTPit will pause your script until it has completed only one file in the queue. If the function returns “Queue has more items.”, then you will know that the queue is not empty and that if you still want to wait for the queue, you will need to call the doFTP-WaitForQueue function again. If the function returns “Queue is now empty.”, then you will know that there are no more items in the queue and you can continue on with your script. The purpose of waiting for only one item in the queue to be finished is so that you can provide your own progress or status on a layout in your database. For example, you could have a loop that calls the doFTP-WaitForQueue with the parameter “WaitFor=One”, and examine the response. If it contains the word “more”, then you could call the [doFTP-GetQueue](#) function to populate a field on the current layout to show your user which files are left in the queue and then loop back up to call doFTP-WaitForQueue

again. If the response contains the word “empty”, then you could exit the loop and disconnect from the server (or start uploading some other files or whatever else you need to do).

You also have the option of bringing up a status dialog that shows the progress of the file or files in the queue. You do this by specifying the “Dialog=Yes” parameter. This parameter will make FTPit show you the normal FTPit Status Dialog while it is pausing your script. (Note that if you have the FTPit Status Dialog already open, it will close it and reopen it for the duration of the pause, and then close it again when it has completed the transfers.) By default, this Status Dialog will have the title “FTPit Progress...” and will say something like “Please wait while FTPit uploads MyFile.pdf”. For those of you who develop custom solutions, you can change this text to include the name of your solution (or any other text) instead of mentioning FTPit. You do this by using the DialogTitle and DialogText parameters as shown in the example below. You will see in the example below that the DialogText is defined as “Please wait while MySolution <<action>> <<filename>>...” If you have worked with merge fields on layouts in FileMaker, then the “<<action>>” and “<<filename>>” strings should look familiar. These work in the same basic way to allow the plug-in to correctly display the current action (uploading or downloading) and the current filename of the file being transferred. So, insert these “merge fields” in the text where it makes sense.

Examples:

```
External("doFTP-WaitForQueue", "")
External("doFTP-WaitForQueue", "WaitFor=One")
External("doFTP-WaitForQueue", "Dialog=Yes")
External("doFTP-WaitForQueue", "Dialog=Yes; DialogTitle=MySolution's Progress; DialogText=Please wait while MySolution <<action>> <<filename>>...")
```

See Also:

[doFTP-CompletedScript](#)
[doFTP-GetQueue](#)
[doFTP-LastCompleted](#)

Understanding Error Responses

Every function in FTPit returns a response indicating the success or failure of that function. If the function is successful, it will return a response indicating that it set the value you were trying to set, completed the task that needed to be completed, or return the information that you requested. If however, the function is not successful, it will return an Error Response. This Error Response is in the form of:

ERROR: <Function Name>: <Error Description>

For example, if you forgot to set a Username using doFTP-Username, and you attempt to connect to the FTP site with [doFTP-Connect](#), the [doFTP-Connect](#) function will return the following Error Response:

ERROR: Connect: Could not connect: No Username specified

Error responses always start with the word "ERROR" in all caps, followed by a colon, followed by the function that had the error, followed by a colon, followed by the actual error that occurred. You can use the various FileMaker Pro Text Functions to extract the different parts of the Error Response for your own use. For instance, if you want to know if the response you just got back was an ERROR, you can use the LeftWords function to return the first word and see if it is an error.

Examples:

```
Set Field ["Result", "External("doFTP-Connect", "")"]
If ["LeftWords(Result, 1) = "ERROR"""]
    <inform the user>
Else
    <upload a file>
End If
```

Credits

Programming, documentation, and example databases by Jake Traynham
Concept, web design, and example databases by Jesse Traynham

Contact Information

Email: FTPit@comm-unity.net
FTPit Website: <http://FTPit.cnsplug-ins.com/>
Main Website: <http://www.cnsplug-ins.com/>
Phone: 817-560-4226
Fax: 817-244-0340

You can write us at:

Comm-Unity Networking Systems
8652 Camp Bowie West
Fort Worth, Texas 76116