

ScriptFTP

The professional tool to automate FTP and secure FTP transfers

Scripting guide and command reference

Carlos Andrés Martínez

Copyright 2004-2008
Carlos Andrés Martínez

Last update: November the 6th, 2008
For the most up to date version visit <http://www.ScriptFTP.com>

INDEX

Getting started	6
Lesson 1: Your first script	7
Lesson 2: Transferring files	9
Lesson 3: Variables	12
Lesson 4: Using IF, WHILE and GOTO	14
Lesson 5: Handling file lists	17
HOW TOs	19
Transferring modified files only	20
Making a backup	23
Logging ScriptFTP messages	28
Sending emails from within a script	30
Error handling	34
ScriptFTP on the command line	38
Updating 1.x scripts	41
Updating 2.x scripts	43
Encrypting script files	46
Operators	47
Arithmetical operators	47
Comparasion operators	48
Logical operators	49
Text value operators	51

Commands for server connection	52
OPENHOST	53
CLOSEHOST	56
SETPROTOCOL	58
SETPORT	61
Commands for file transfer	62
GETFILE	63
PUTFILE	65
SYNC	67
ADDEXCLUSION	71
CLEAREXCLUSION	73
SETTYPE	74
SETPASSIVE	76
SETSPEED	80
SETCLOCKDIFF	81
SETUPLOADMODE	82
Commands for directory handling	84
LOCALCHDIR	85
LOCALCWDIR	87
LOCALMKDIR	89
LOCALRMDIR	90
CHDIR	91
CWDIR	92
MKDIR	94

RMDIR	96
Commands for file handling	97
RENAMEFILE	98
DELETEFILE	100
CHMOD	102
Handling local files	104
Hint: Moving remote files	106
Script output commands	109
PRINT	110
SILENT	112
VERBOSE	115
LOGTO	117
Miscellaneous commands	119
GETLIST	120
GETDATE	123
GETENV	125
GETPARAM	126
RAWCOMMAND	128
EXEC	129
SLEEP	131
STOP	132
EXIT	134

TEXCUT	136
TEXTLENGTH	138
Advanced topics and rare features	139
License Agreement	142

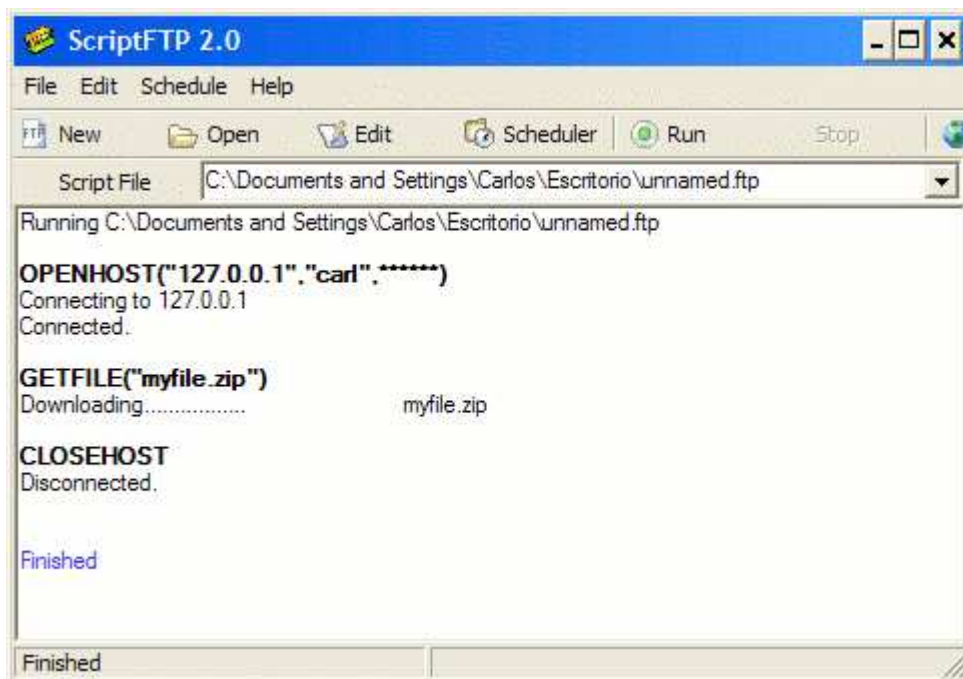
Getting started

Lesson 1: Your first script

ScriptFTP is a script-driven FTP client. It works like traditional FTP clients but does not require any user interaction while running, instead it works automatically using a text file which contains the actions ScriptFTP has to execute. Let us consider a simple script:

```
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")  
GETFILE ("myfile.zip")  
CLOSEHOST
```

Click on File → New → Empty Script, paste the above script into the editor window and replace "ftp.myhost.com", "myuser" and "mypassword" with your settings. Then click RUN in the ScriptFTP window. Do not forget to save your modifications in the editor window first.



As you can see this script connects to the FTP server, downloads a file and then disconnects. It is a fairly simple script.

In order to familiarize yourself with the commands supported by ScriptFTP take a look at the command list. You may also take a look at [GETFILE](#), [PUTFILE](#) or [OPENHOST](#).

Lesson 1 has finished now. However, before moving on to the next lesson take a look at the [script examples](#) section on the ScriptFTP homepage. There you will find a collection of scripts for various purposes.

Lesson 2: Transferring files

The commands [GETFILE](#), [PUTFILE](#) and [SYNC](#) are used for file transfers. Since the purpose of ScriptFTP is transferring files, it is important to understand the use of these commands before you write your own script.

The [GETFILE](#) command is used for downloading a set of files from the FTP server. It has two parameters - the first one indicates the file or files you want to download. If you also need to download files from subdirectories add the second parameter: SUBDIRS.

```
# Connect to server  
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")  
  
# Download a.jpg  
GETFILE ("a.jpg")  
  
# Download every file in cgi-bin  
GETFILE ("cgi-bin/*.*")  
  
# Download backup-2005-12-3.zip from backup  
GETFILE ("/backup/backup-2005-12-3.zip")  
  
# Download all files in the directory images  
GETFILE ("/images/*.*", SUBDIRS)  
  
# Close connection  
CLOSEHOST
```

Note that when using the SUBDIRS parameter ScriptFTP will preserve the original directory structure and will therefore create the appropriate directories on the local drive.

Other commonly used script commands are [CHDIR](#) and [LOCALCHDIR](#). The first one sets the current remote directory and the second one is used to set the current local directory. ScriptFTP will download all files to the current local directory so you had better use [LOCALCHDIR](#) before calling [GETFILE](#). Let us look at an example demonstrating the use of both commands:

```

# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Set current local directory
LOCALCHDIR ("C:\dest_dir")

# Set current remote directory
CHDIR ("/images")

# Download all files from /images to
# the directory images in C:\dest_dir
GETFILE ("*.*", SUBDIRS)

# Close connection
CLOSEHOST

```

The **PUTFILE** command obeys the same syntax. Let us look at an example:

```

# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Set current local directory
LOCALCHDIR ("C:\orig_dir")

# Set current remote directory
CHDIR ("/images")

# Upload all files from C:\orig_dir to
# /images
PUTFILE ("*.*", SUBDIRS)

# Close connection
CLOSEHOST

```

And finally the **SYNC** command. Use this command for synchronizing directories. In ScriptFTP the term synchronization means "Get an exact copy of a directory transferring new and modified files only. Delete unwanted files.". This definition of synchronization may sound complex, but you will understand it as soon as you see the **SYNC** command in action:

```
# Connect to server  
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")  
  
# Synchronize C:\dest_dir from /www  
SYNC ("C:\dest_dir", "/www", DOWNLOAD)  
  
# Close connection  
CLOSEHOST
```

The first parameter indicates a local directory, the second one a remote directory and the third one the direction of the synchronization. You can synchronize both a local directory with a remote directory (DOWNLOAD) and a remote directory with a local one (UPLOAD). For example, if you want to publish a web site the command is: SYNC("C:\local_webdir", "/www", UPLOAD)

The [SYNC](#) command also has a fourth parameter which is optional: SUBDIRS. Its meaning is the same as in [GETFILE](#) or [PUTFILE](#).

For further information take a look at the commands help reference: [GETFILE](#), [PUTFILE](#), [SYNC](#). Also see [Transferring modified files only](#).

Lesson 3: Variables

If you have done some programming before you will probably be familiar with the term "variable". However, this tutorial is intended to teach the concept of variables from the ground up. So here is a definition: "A variable is a language item to store something". In ScriptFTP that something means just text. Let us look at an example:

```
# Store the text Hello world in myvariable  
$myvariable="Hello world"  
  
# Print Hello world in the ScriptFTP window  
PRINT ($myvariable)
```

The output of the script is the following:

```
Hello world
```

A variable does not need to be declared before use. In other words, you do not need to state explicitly in your script that the word "my_number" is a variable you are going to use. Just use the variable. The only restriction on variable names is that command names are not allowed.

You can use variables for storing the parameters of a command and then call the command using these variables.

```
$host="ftp.myhost.com"  
$user="myuser"  
$password="123456"  
  
OPENHOST ($host, $user, $password)  
PUTFILE ("*.*", SUBDIRS)  
CLOSEHOST
```

Variables may also be used to store the value a command returns:

```
# Connect to server and store the  
# return value of OPENHOST in $result
```

```
$result=OPENHOST (host,user,password)

# If the result is "OK" transfer the
# files and disconnect
IF ($result=="OK")
    PUTFILE ("*.*",SUBDIRS)
    CLOSEHOST
END IF
```

Just like commands ScriptFTP features a set of operators for calculating arithmetical or logical expressions. In the above example we used the == operator in order to check whether the content of the variable is equal to "OK". Operators also include +,-,*,/ among others. Have a look at [Operators](#) for more information.

The use of IF is also described in the [next section](#). See also [Error handling](#).

As mentioned before every ScriptFTP variable holds a text string and therefore it is necessary to enclose the text with quotes, however, if you supply a number, the quotes may be omitted.

```
$myvariable=1
PRINT ($myvariable)
PRINT ("2")
PRINT (-3)
```

The script output is:

```
1
2
-3
```

Lesson 4: Using IF, WHILE and GOTO

IF and WHILE are language elements that evaluate conditions. You may use them for evaluating error conditions, performing actions a fixed number of times etc. Since every variable in ScriptFTP contains text, the text "TRUE" means true. Any other text is evaluated to false. For example:

```
IF ("TRUE")
    PRINT ("ScriptFTP rules")
END IF

IF ("fdasfdas")
    PRINT ("hi")
END IF

$myvariable="TRUE"

if ($myvariable)
    PRINT ("ScriptFTP rules")
END IF
```

And the output is:

```
ScriptFTP rules
ScriptFTP rules
```

As you can see from the script output the IF statement only executes its body if the condition equals the text "TRUE".

Let us look at a more useful example. In the following script the IF statement is used to check whether [OPENHOST](#) was successful:

```
# Connect to ftp.myhost.com and download
# all files in the www folder.
#
#
# The server and login are:
$host="ftp.myhost.com"
```

```

$user="carl"
$password="1234567890"

# This is a label, it marks a point
# in the script file
:start

# Connect to FTP server
$result=openhost ($host, $user, $password)

# check what happened with openhost
IF ($result=="OK")
    PRINT ("CONNECTED")
ELSE
    PRINT ("Cannot connect. Waiting 5 seconds.")
    SLEEP (5)
    GOTO :start
END IF

# do the stuff
LOCALCHDIR ("C:\localwww\")
CHDIR ("www")
GETFILE ("*.*")

CLOSEHOST ()
PRINT ("FINISHED")
EXIT

```

The WHILE statement will execute the commands in its body as long as the condition equals "TRUE". In the following script the WHILE statement will never terminate and will therefore keep a local directory synchronized with an FTP site:

```

# This script will loop indefinitely and always keep a
# local directory synchronized with an FTP site.
#
#
# Server and login information
$host="172.16.0.4"
$user="carl"
$password="123456"

```

```
# Connect to server
OPENHOST($host,$user,$password)

# Change current remote directory
CHDIR("remotedir/remotesubdir")

# Change current local directory
LOCALCHDIR("C:\localdir\")

# Loop indefinitely
WHILE ("TRUE")
    $result=GETFILE ("*.*)
    IF ($result!="OK")
        PRINT ("ERROR in GetFile, exiting")
        # Exit with error code 1
        EXIT (1)
    ELSE
        PRINT ("All files downloaded. Waiting 10 seconds")
        SLEEP (10)
    END IF
END WHILE
```

Lesson 5: Handling file lists

Sometimes it is useful to perform different actions for each item in a set of files. For example, if you do the following:

```
# Connect to server
OPENHOST ("ftp.myhost.com", "joe", "123456")

# Download all files
GETFILE (*.*)

# Delete the downloaded files
DELETEFILE (*.*)

# Close the connection
CLOSEHOST
```

It may happen that [GETFILE](#) fails and remote files will be deleted although they have never been downloaded. In order to avoid this issue we will retrieve a file listing with the [GETLIST](#) command, call [GETFILE](#) for each file in the listing and delete this remote file using [DELETEFILE](#) if no error has occurred.

```
# Connect to server
OPENHOST ("ftp.myhost.com", "joe", "123456")

# Change the current local directory. All files
# will be downloaded here.
LOCALCHDIR ("C:\users\carlos\desktop\localftp")

# Get the remote file listing, store it in $list
GETLIST ($list, REMOTE_FILES)

# For each file in $list...
FOREACH $item IN $list
    # Download the file
    $result=GETFILE ($item)

    # If the file has been downloaded successfully
    # delete the remote copy. Else stop the script.
```

```
    IF ($result=="OK")
        DELETEFILE ($item)
    ELSE
        STOP
    END IF
END FOREACH

# Close the connection
CLOSEHOST
```

[GETLIST](#) commands are usually followed by FOREACH loops. See [GETLIST](#) for further reference about parameters and options.

HOW TOs

Transferring modified files only

Synchronization is one of the most complex operations ScriptFTP can carry out. This is because the command used for this purpose ([SYNC](#)) may sometimes appear complex and confusing. It is important to understand that in ScriptFTP synchronization can only be a one-way operation. This means that depending on the SYNC method you choose ScriptFTP will upload new and modified local files to a remote directory *or* download new and modified remote files to a local directory. These tasks are mutually exclusive. Optionally SYNC can also delete orphaned files. Before you start wondering about the meaning of "orphaned" let us look at a very simple example:

```
# Connect to ftp.host.com as mysuer
OPENHOST ("ftp.host.com", "myuser", "mypassword")

# Upload new and modified files from /www
# to C:\LocalWebFolder. Delete orphaned files
SYNC ("C:\LocalWebFolder", "/www", UPLOAD_DELETE)

# Close the connection
CLOSEHOST
```

Once you have clicked the Run button ScriptFTP will display the following messages:

```
OPENHOST ("ftp.host.com", "myuser", *****)
Connecting to ftp.host.com
Connected.

SYNC ("C:\LocalWebFolder", "/www", UPLOAD_DELETE)
Uploading temporary file to calculate server-client clock time
difference.
Clock time difference is -7201 seconds.
Synchronizing remote directory /www from C:\LocalWebFolder
Deleting      (remote file is orphaned)      orphaned_file.txt
Uploading     (remote file not found)        about.html
Skipping      (remote file is uptodate)      back.jpg
Uploading     (remote file is older)         contact.html
Uploading     (remote file not found)        index.html
Skipping      (remote file is uptodate)      notes.txt
```

CLOSEHOST

Disconnected.

The remote file orphaned_file.txt has been deleted because it could not be found in C:\LocalWebFolder, this is why it is called an orphaned file. The next file about.html has been uploaded because it did not exist in the remote directory /www. When a file is found in both remote and local locations ScriptFTP can perform two actions: retransfer it or simply ignore it (skip). The choice of action depends on the file modification date. If the file is newer ScriptFTP will transfer it.

As you can see a synchronization can take three different actions for each file: transfer, delete or skip. Thus ScriptFTP will achieve an exact copy of a local or remote directory by transferring only the files needed. The method by which ScriptFTP finds all modified files is comparing local and remote file modification time stamps. If the computer ScriptFTP is running on and the FTP server share exactly the same time (which usually never happens) ScriptFTP will only have to compare the time stamps of the files. In the real world with computers spread around the world having different local times, it is more complex to determine whether a remote file is older or newer than a given local one. In order to solve this time difference problem ScriptFTP has the ability to determine the time difference between server and client. This is what you will notice:

```
(...)  
Uploading temporary file to calculate server-client clock time  
difference.  
Clock time difference is 3684 seconds.  
(...)
```

3684 seconds comprise approximately one hour. This is the server-client time difference. ScriptFTP will use this value to determine which file is older and will transfer it only if necessary.

Sometimes it may happen that ScriptFTP cannot determine this value. This is usually due to access restrictions. For example:

```
Uploading temporary file to calculate server-client clock time  
difference.  
***** SYNC Error 550: Cannot upload temporary file.  
***** The server said: Permission denied
```

In this case you will have to manually tell ScriptFTP the time difference using the [SETCLOCKDIFF](#) command:

```
SETCLOCKDIFF (-3600)  
  
OPENHOST ("ftp.host.com", "myuser", "mypassword")  
SYNC ("C:\LocalWebFolder", "/www", UPLOAD_DELETE)  
CLOSEHOST
```

For further information see [SETCLOCKDIFF](#) and [SYNC](#).

Making a backup

This section will show you how to make a backup of a local directory. The backup will be performed by creating a ZIP file of the local directory and then uploading it to an FTP site.

We will use the [EXEC](#) command for creating the ZIP file. This command is used to call external programs from within the script. In the following examples the [EXEC](#) command will call the free command-line tool [Info-Zip](#) to create the ZIP file. However, you may use any other command-line ZIP utility, for example, [PkZip](#) or [PowerArchiver](#). We will use the [EXEC](#) command like this:

```
EXEC("C:\Info-Zip\zip.exe -r MyBackupFile.zip C:\MyFolder")
```

When ScriptFTP reaches this line it will call C:\Info-Zip\zip.exe and wait for the external program to finish. Once zip.exe has finished we will have MyBackupFile.zip with the contents of C:\MyFolder in the local working directory (change it with [LOCALCHDIR](#)) and will be ready to upload it to the FTP site:

```
# Create the backup file  
EXEC("C:\Info-Zip\zip.exe -r MyBackupFile.zip C:\MyFolder")  
  
# Connect to server  
OPENHOST("ftp.myhost.com", "myuser", "mypass")  
  
# Upload the file  
PUTFILE("MyBackupFile.zip")  
  
# Close host  
CLOSEHOST
```

This method will replace the zip file everytime it is uploaded to the FTP site. If you want to avoid this problem we can add the current date to the ZIP filename:

```
# Get the current date  
\$current\_date=GETDATE(FORMAT3)  
  
# Build the zip filename
```

```

# concatenating text strings
$zip_file_name="MyBackupFile-".$current_date.".zip"

# Build the command line
$command_line="C:\Info-Zip\zip.exe -r ".$zip_file_name."
C:\MyFolder"

# Create the zip file
EXEC($command_line)

# Connect to server
OPENHOST("ftp.myhost.com","myuser","mypass")

# Upload the file
PUTFILE($zip_file_name)

# Close connection
CLOSEHOST

```

You may add even more features to this script file. The following script will make the backup and also delete the intermediate zip file after successful upload. It will also log the script run and it includes some error handling.

```

# ----- Enter your own settings here -----
# FTP server
$ftp_server="127.0.0.1"
$ftp_user="carl"
$ftp_password="123456"

# The local directory you need to backup
$backup_dir="C:\MyDir\"

# The ZIP file name
$zip_file_name="MyBackup.zip"

# The remote destination folder where
# the ZIP file is uploaded to
$remote_destination_folder="/"

# Get the Windows temp directory from
# the system environment variable TEMP

```

```

# The backup ZIP file will be created there.
$local_temp_dir=GETENV("TEMP")

# The path of the free tool Info-zip
# used to create the zip file
$info_zip_path="C:\infozip\"

# -----Script starts here -----

# Get the current date
$current_date=GETDATE(FORMAT3)

# Append the current date to the ZIP file name
# ( -> YYYYMMDD_MyBackup.zip)
$zip_file_name=$current_date."_".$zip_file_name

# Build the command line used to call the external ZIP tool
$backup_command_line=$info_zip_path."\zip.exe -r
".$zip_file_name." ".$backup_dir

# Go to the temp directory where the ZIP file will be created
$result=LOCALCHDIR($local_temp_dir)
IF($result!="OK")
    PRINT("Error changing current local directory. Closing
ScriptFTP in 5 seconds")
    SLEEP(5)
    EXIT($result)
END IF

# Run Info-zip archiver
$result=EXEC($backup_command_line)

# Check if the ZIP file has been created
IF($result!=0)
    PRINT("Error. The ZIP file could not be created. Closing
ScriptFTP in 5 seconds")
    SLEEP(5)
    EXIT($result)

```

```

END IF

# Once this point is reached the ZIP file will already be created

# Connect to FTP server
$result=OPENHOST ($ftp_server, $ftp_user, $ftp_password)
IF ($result!="OK")
    PRINT ("Error. Cannot connect to FTP server. Closing
ScriptFTP in 5 seconds")
    SLEEP (5)
    EXIT ($result)
END IF

# Change current remote dir to the destination dir
$result=CHDIR ($remote_destination_folder)
IF ($result!="OK")
    PRINT ("Error. Cannot go to remote destination dir. Closing
ScriptFTP in 5 seconds")
    SLEEP (5)
    CLOSEHOST
    EXIT ($result)
END IF

# Upload the ZIP file
$result=PUTFILE ($zip_file_name)
IF ($result!="OK")
    PRINT ("Error. Cannot upload the ZIP file. Closing ScriptFTP
in 5 seconds")
    SLEEP (5)
    CLOSEHOST
    EXIT ($result)
END IF

CLOSEHOST

# Delete the local copy of the zip file
EXEC ("del /F /Q ".$zip_file_name)

PRINT ("-----")

```

```
PRINT ("Backup finished. Closing ScriptFTP in 5 seconds.")  
PRINT ("-----")  
  
SLEEP (5)  
EXIT (0)
```

Logging ScriptFTP messages

The [LOGTO](#) command is used to log ScriptFTP output to a text file. After calling this command, every message shown on the ScriptFTP window will be added to the specified text file (log file). With this feature you can check if your scheduled job has been carried out correctly.

Let us look at an example:

```
# Log output to C:\transfer_log.txt
LOGTO ("C:\transfer_log.txt")

# Connect to server
OPENHOST ("ftp.host.com", "myuser", "mypassword")

# Download some files
GETFILE ("ClientDocs/*.*", SUBDIRS)

# Close connection
CLOSEHOST ()

# Close ScriptFTP. The script output is saved
# to the log file so we do not need to keep the
# ScriptFTP window opened.
EXIT
```

In the above example ScriptFTP overwrites the log file each time it is executed. If you want to keep the history of subsequent runs you may append the current date to the log file name:

```
# Build the log file name and path using the current date
$log_file_name="C:\log_file_name-".GETDATE (FORMAT2) ".txt"

# Log script output
LOGTO ($log_file_name)

# Connect to server
OPENHOST ("ftp.host.com", "myuser", "mypassword")
```

```
# Download some files
GETFILE ("ClientDocs/*.*", SUBDIRS)

# Close connection
CLOSEHOST ()

# Close ScriptFTP. The script output is saved
# to the log file so we do not need to keep the
# ScriptFTP window opened.
EXIT
```

You may also supply the optional parameter APPEND to the [LOGTO](#) command. Thus ScriptFTP will append its output to the log file each time the script is executed:

```
# Log output to C:\transfer_log.txt
LOGTO ("C:\transfer_log.txt", APPEND)

# Connect to server
OPENHOST ("ftp.host.com", "myuser", "mypassword")

# Download some files
GETFILE ("ClientDocs/*.*", SUBDIRS)

# Close connection
CLOSEHOST ()

# Close ScriptFTP. The script output is saved
# to the log file so we do not need to keep the
# ScriptFTP window opened.
EXIT
```

As a hint, if you find the script's output too detailed use the [SILENT](#) command to have ScriptFTP display the essential information only.

Sending emails from within a script

ScriptFTP has been designed for unattended operation and usually you will know what has happened during a script run just by taking a look at the ScriptFTP window. Additionally, you may [log](#) a script's output messages to a text file and even have it send emails to you which contain information about the run. This section is dedicated to this topic.

In ScriptFTP there is no built-in command for sending emails because it is not needed. We will rather use the [EXEC](#) command to call an external command-line program that will send the email. The following examples use [Blat](#), a free and tremendously useful command-line mail program for all kinds of batch jobs. However, you can use any other mail program just by adjusting the [EXEC](#) call.

The following example script will send an email if it cannot connect to the FTP server or if the file synchronization fails. Note that we first setup the way how Blat will be invoked. We construct two different commands for calling Blat, one for each type of error, since the user should tell from the email what happened exactly and not be confused by a generic error message.

```
# FTP server settings
$ftp_server="ftp.myserver.com"
$ftp_user="myuser"
$ftp_password="mypass"

# Set the directories that this script
# will synchronize.
$remote_dir_to_synchronize="/remotedir/"
$local_dir_to_synchronize="C:\MyDir"

# Blat parameters. This is the command line program
# used in this example to send emails from within ScriptFTP

$blat_path="C:\blat\blat.exe"

$smtp_server="smtp.myserver.com"
$smtp_user="myuser_at_myserver.com"
$smtp_password="mypassword"
```

```

$email_from="scriptftp@myserver.com"
$email_to="me@myserver"
$email_subject="ScriptFTP error message"
$email_body1="Could not connect to FTP server"
$email_body2="Could not synchronize files"

# Build the log file path by retrieving Windows' temp path from
# the system environment variable TEMP and appending the current
# date to the file name "logfile-". For Example:
# C:\windows\temp\logfile-20070424.txt
$log_file_path=GETENV("TEMP")."\logfile-".GETDATE(FORMAT3)".txt"

# Start logging the script's output
LOGTO($log_file_path)

# The blat command for sending an email is supposed
# to look like this (without the line breaks):
#
# C:\blat\blat.exe
# -server smtp.myserver.com
# -u myuser_at_myserver.com
# -pw mypassword
# -f scriptftp@myserver.com
# -to me@myserver
# -subject "ScriptFTP error message"
# -body "Error message here"
# -ps "C:\Users\Carlos\AppData\Local\Temp\logfile-20070427.txt"
#
# As there are two different kinds of emails depending
# on the errors that may have happened we need to build
# two different blat commands
#
# Both have a common part which consists of the variables
# $common_part_1, $common_part_2 and $common_part_3.
# The fourth part contains the different body and
# they are called $cmd_line_part_4_1 and $cmd_line_part_4_2
#
$common_part_1=$blat_path." -server ".$smtp_server." -u

```

```

".$smtp_user." -pw "
$common_part_2=$smtp_password." -f ".$email_from." -to
".$email_to." -subject ".'"
$common_part_3=$email_subject.'"

$cmd_line_part_4_1=" -body ".'"'.$email_body1.'".'" -ps
".'"'.$log_file_path.'"
$cmd_line_part_4_2=" -body ".'"'.$email_body2.'".'" -ps
".'"'.$log_file_path.'"

# Concatenate the text strings to build the complete commands
$blat_cmd_line_1=$common_part_1.$common_part_2.$common_part_3.$cmd_line_part_4_1
$blat_cmd_line_2=$common_part_1.$common_part_2.$common_part_3.$cmd_line_part_4_2

# Connect to the FTP server
$result=OPENHOST ($ftp_server,$ftp_user,$ftp_password)
IF($result!="OK")
    PRINT("Cannot connect to FTP server. Sending an email and aborting in 5 seconds.")
    EXEC($blat_cmd_line_1)
    SLEEP(5)
    EXIT(1)
END IF

# Synchronize the files
$result=SYNC($remote_dir_to_synchronize,$remote_dir_to_synchronize,UPLOAD)
IF($result!="OK")
    PRINT("Cannot synchronize. Sending an email and aborting in 5 seconds.")
    EXEC($blat_cmd_line_2)
    SLEEP(5)
    EXIT(2)
END IF
CLOSEHOST

```

EXIT (0)

Error handling

Every ScriptFTP command returns a text string. If everything goes well during the execution of the command it will return the text "OK", if something goes wrong you will get an error code. Evaluating this return value and taking the appropriate measures you can make your file transfers fault-tolerant.

In the following example the output of [OPENHOST](#) is stored in a variable called *\$result*. If *\$result* is "OK" we will continue, if *\$result* is different from "OK" we will show a message and try to reconnect to the FTP server.

```
# This is a label. It marks a point in the script.
# We will use it to return to this point if a
# connection attempt fails.
:connect

# Shows a message
PRINT ("____Connecting____")

# Connect to server. The return value of OPENHOST
# is stored in $result
$result=OPENHOST ("myserver.com", "me", "13579")

# Check if $result is different from "OK"
IF ($result!="OK")
    PRINT ("Cannot connect! Trying again.")
    # Jump to the label :connect to retry
    # the connection
    GOTO :connect
END IF

# Once this point is reached ScriptFTP
# will be connected to the server.
# Transfer the files.
GETFILE ("*.*")

# Close connection
CLOSEHOST
```

The example above will try to connect to the FTP server indefinitely. Let us add some code in order to make only three attempts:

```
# This variable will store the connection attempts done.
# It is initially is set to 0.
$attempts=0

# This is a label. It marks a point in the script.
# We will use it to return to this point if a
# connection attempt fails.
:connect

# Add 1 to the connection attempts counter
$attempts=$attempts+1

# Display a message
PRINT("Connecting. Attempt number ".$attempts)

# Connect to server. The return value of OPENHOST
# is stored in $result
$result=OPENHOST("myserver.com","me","13579")

# Check if $result is different from "OK"
IF($result!="OK")
    # If this is the third attempt stop execution
    IF($attempts==3)
        STOP
    ELSE
        PRINT("Cannot connect! Trying again.")
        # Jump to the label :connect to retry
        # the connection
        GOTO :connect
    END IF
END IF

# Once this point is reached ScriptFTP
# is connected to the server.
# Transfer the files.
GETFILE("*.*)"
```

```
# Close connection  
CLOSEHOST
```

The command [PUTFILE](#) also outputs a return value. We will evaluate it in order to check whether the upload has been performed successfully:

```
$webserver="www.whatever.com"  
$myuser="me"  
$mypassword="13579"  
  
OPENHOST ($webserver, $myuser, $mypassword)  
  
$my_result_put=PUTFILE ("*.*)"   
  
# If PUTFILE returns anything different  
# from OK jump to :failed_put  
IF ($my_result_put!="OK")  
    GOTO :failed_put  
END IF  
  
:allright  
PRINT ("All right")  
  
# Close connection  
CLOSEHOST ()  
  
# Wait 60 seconds  
SLEEP (60)  
  
# Close ScriptFTP  
EXIT  
  
:failed_put  
PRINT ("Error found putting files")  
  
# Close connection  
CLOSEHOST ()  
  
# Wait 60 seconds
```

```
SLEEP (60)
```

```
# Close ScriptFTP
```

```
EXIT
```

You can find more examples of error handling in the examples section of the website. The advanced scripts therein cover this issue in detail.

ScriptFTP on the command line

ScriptFTP usually runs in a window but there is also a command line version of it. It works exactly the same way as the usual ScriptFTP except that it will not display any windows, it will just write the script messages to the command prompt. The program file is called **scriptftp_console.exe** and you can find it in the directory where you installed ScriptFTP. As this command prompt version does not display any windows or dialog boxes you must specify which script you want to run:

```
ScriptFTP_console.exe <script_file_path> [custom parameter]
[custom parameter]....[custom parameter]
```

As usual in the command line tools, the symbols < and > indicate a mandatory parameter whereas [and] are used for optional parameters.

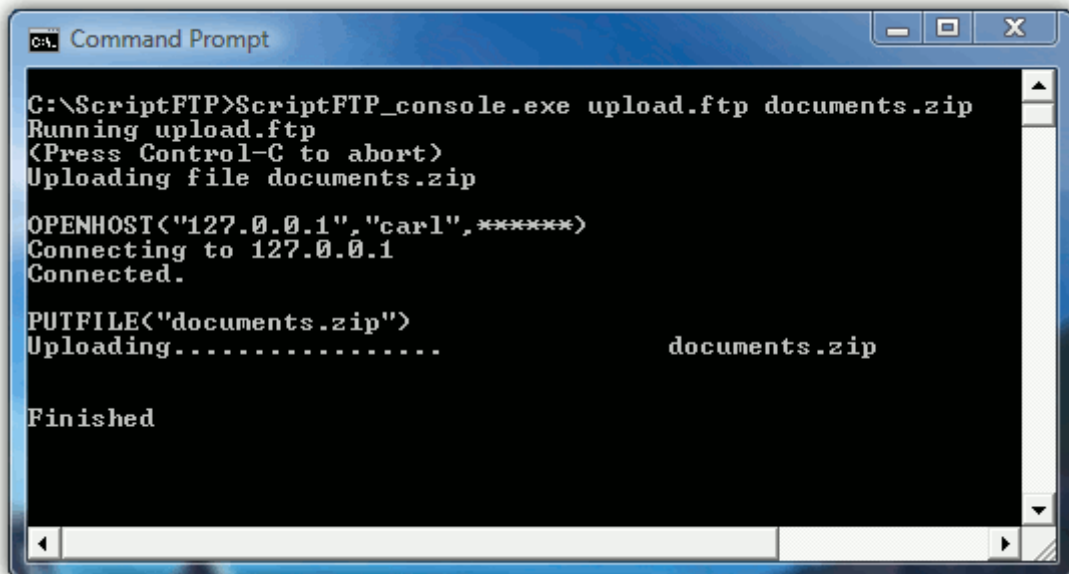
For example:

```
ScriptFTP_console.exe C:\MyScripts\upload.ftp
```

If the script path contains spaces remember to use double quotes to enclose it:

```
ScriptFTP_console.exe "C:\My Scripts\upload some files.ftp"
```

The custom parameters are optional and can be used in order to pass information to your script, for example, the FTP login or the files to be uploaded. You can access these parameters from within your script file using the [GETPARAM](#) command. Note that ScriptFTP.exe (the GUI version of ScriptFTP) obeys the same syntax and also accepts command line parameters.



```
C:\> Command Prompt
C:\ScriptFTP>ScriptFTP_console.exe upload.ftp documents.zip
Running upload.ftp
<Press Control-C to abort>
Uploading file documents.zip

OPENHOST<"127.0.0.1","carl",*****>
Connecting to 127.0.0.1
Connected.

PUTFILE<"documents.zip">
Uploading..... documents.zip

Finished
```

```
# This script is used for uploading the specified file
# to the FTP server. Use it this way:
# ScriptFTP.exe this_script.ftp the_file_to_upload

# Note:
# The GETPARAM command was added in ScriptFTP 2.1
# Build March 14th 2006.

$param3 = GETPARAM(3)

PRINT("Uploading file ".$param3)

OPENHOST("127.0.0.1","carl","123456")

PUTFILE($param3)

CLOSEHOST
```

If you need to get ScriptFTP completely hidden in the background use ScriptFTP.exe (not scriptftp_console.exe) with /HIDE as the second parameter:

```
ScriptFTP.exe background_script.ftp /HIDE
```

You can even start ScriptFTP.exe (not scriptftp_console.exe) as an icon in the system tray:



```
ScriptFTP.exe run_minimized_in_the_tray.ftp /TRAY
```

If you want ScriptFTP.exe to close itself once the script has finished use the [EXIT](#) command in your script or the /AUTOCLOSE command line parameter:

```
ScriptFTP.exe closes_itself_once_finished.ftp /AUTOCLOSE
```

Updating 1.x scripts

The 3.x script language is identical to the 1.x version except from the following minor differences:

- Every variable must start with the character "\$". Put \$ at the beginning of every variable.
- Every label must start with the character ":". Put : at the beginning of every label.
- The commands ISEQUAL, NOT, ADD and CONCAT are still supported but they are deprecated. As the 3.x script language supports [operators](#) you might want to replace them with their corresponding arithmetical symbols.
- Some commands have changed their names and now the synchronization feature has been turned into a separate command ([SYNC](#)). See the table below for command correspondence.

Old command name	New name	Syntax changes
FTPOPENHOST	OPENHOST	no changes
FTPCLOSEHOST	CLOSEHOST	no changes
FTPGETFILE	GETFILE	Synchronization/mirror feature has been moved to the new SYNC command
FTPPUTFILE	PUTFILE	Synchronization/mirror feature has been moved to the new SYNC command
TRANSFERMODE	SETTYPE	Same syntax. Added EBCDIC.
LOCALCWD	LOCALCWDIR	no changes.
FTPCHDIR	CHDIR	no changes.
FTPCWD	CWDIR	no changes.
FTPMKDIR	MKDIR	no changes

FTPRMDIR	RMDIR	no changes.
FTPFILERENAME	RENAMEFILE	no changes.
FTPFILEDEL	DELETEFILE	no changes.
FTPLS	GETLIST	no changes.
FTPCHMOD	CHMOD	no changes.

All commands not shown in this list are not affected by any changes.

Updating 2.x scripts

Changes in the script language compared to its 2.x version are minimal, but it is required to change bits and pieces of a script file in order to have it working in the 3.x series:

- **Every variable must start with the character "\$".** Put \$ at the beginning of every variable. See the example below.

```
# Variables:
#
# 2.x:
myuser="joe"
mypass="1234"
myserver="ftp.host.com"
myresult=OPENHOST(myserver,myuser,mypass)

# 3.x
$myuser="joe"
$mypass="1234"
$myserver="ftp.host.com"
$myresult=OPENHOST($myserver,$myuser,$mypass)
```

- **Every label must start with the character ":".** Put : at the beginning of every label. See the example below.

```
# Labels:
#
# 2.x
:mylabel
GOTO mylabel

# 3.x
:mylabel
GOTO :mylabel
```

- **The commands ISEQUAL, NOT, ADD and CONCAT are no longer supported.** You have to replace them with their [corresponding symbol](#). See the example below:

```
# ADD, NOT, ISEQUAL and CONCAT
#
# 2.x
num=ADD (num, 3)

IF (NOT (ISEQUAL (result, "OK")))
    PRINT ("operation failed")
END IF

IF (ISEQUAL (result, "12451"))
    PRINT ("Access error")
END IF

mymessage=CONCAT ("hello ", name)
PRINT (mymessage)

# 3.x
$num=$num+3

IF ($result!="OK")
    PRINT ("operation failed")
END IF

IF ($result==12451)
    PRINT ("Access error")
END IF

$mymessage="hello ".$name
PRINT ($mymessage)
```

- **The command LIST is no longer supported.** Use GETLIST and FOREACH instead. See the example below:

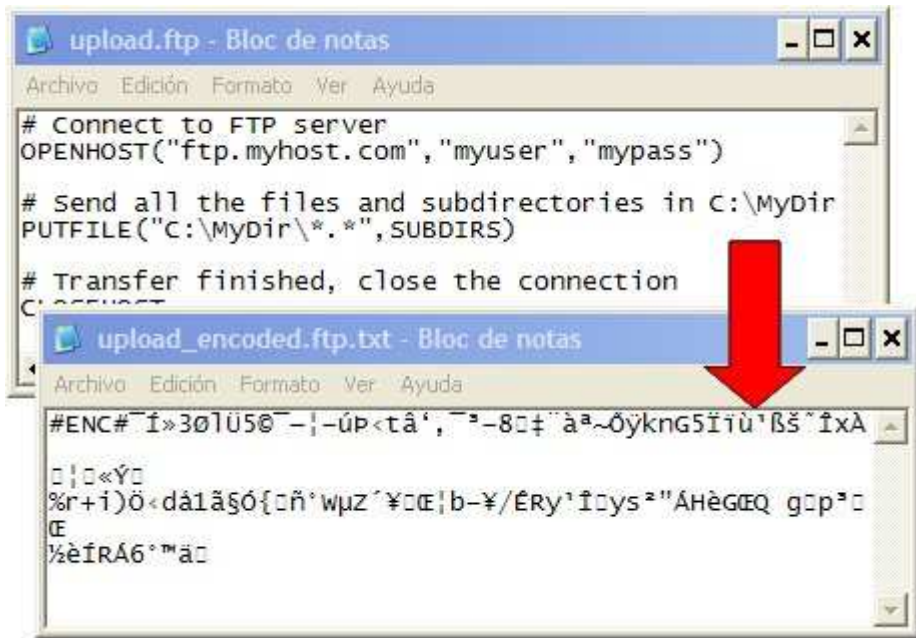
```
# ScriptFTP 2.x:
remote_file_listing=LIST()
```

```
PRINT (remote_file_listing)

# ScriptFTP 3.x:
GETLIST ($list, REMOTE_FILES)
FOREACH $item IN $list
    PRINT ($item)
END FOREACH
```

- **The command SETSSL has been renamed to SETPROTOCOL.** The syntax and parameters are the same.

Encrypting script files



Script files may contain sensitive information such as passwords, user names or remote server information. When ScriptFTP is deployed on many workstations it is advisable to encrypt the script files in order to prevent users from reading the script content.

For encrypting a script file you will have to ask for a small tool at scriptftp@scriptftp.com. The encryption process will leave the file name and extension unaltered and renders the content of the script file unreadable. ScriptFTP will treat the encrypted file as a normal script file.

This feature was added in ScriptFTP 2.1 build March 14th 2006

Operators

Operators are ScriptFTP symbols used for performing arithmetical or logical operations, there are also operators for comparison and text string concatenation. The behavior of operators, like in any other programming language, is very similar to commands. They usually take two values or variables and return one. The operators supported by ScriptFTP are the following:

Arithmetical operators

- + Sum.
- Subtract.
- * Multiply.
- / Divide.

```
# Assign 32 to the $A variable
$A="32"

# Assign 1 to the $B variable
$B="1"

# $C is the sum of $A and $B
$C=$a+$B

# $D is the result of subtracting 1 to $A
$D=$A-1

# Multiplying 5 and 7 you get $E
$E=5*7

# The use of parenthesis is also allowed:
$F=( $A+(100/$C)-2)*5

# If you try to use a text value in an arithmetical
# expression it's interpreted as a zero:
$thisiszero= 5*"abcdef"
```

As the ScriptFTP language is intended for batch operations and automatic transfers, more complex operations such as square roots or powers are not supported.

Comparasion operators

- < Less than.
- > Greater than.
- <= Less or equal than.
- >= Greater or equal than.
- == Equal.
- != Not equal.

The ScriptFTP language is typeless. There are no data types and every value is stored as a text string, even numbers. Every logical operation like the [IF statement](#) or the logical operators below rely on the interpretation of true as the text "TRUE" and false as "FALSE". This means that the logical operators will indeed return a text value containing "TRUE" or "FALSE". Let's see an example:

```
# Store 4 in the variable $A
$A=4

# Store 3 in the variable $B
$B=3

# Is $B greater than $A? Store the result in $R
$R= $B>$A

# Show the content of $R in the ScriptFTP window
# FALSE should be printed
PRINT ($R)

# If $B was greater than $A show a message, else show other
message
IF ($R)
    PRINT ("The value stored in $B is greater than the one
stored in $A")
ELSE
```

```

        PRINT ("The value stored in $B is not greater than the one
stored in $A")
END IF

# You should see the second message

```

Logical operators

AND Logical AND operation.

OR Logical OR operation.

The logical operators returns "TRUE" or "FALSE", this is because, as stated before, ScriptFTP language is typeless. Every value is stored as text and the logical values true or false are stored as "TRUE" or "FALSE" respectively.

```

# Assign some values:
$A=-5
$mytext="abcdef"

# If $a is less or equal than 4 and $mytext is equal to "abcdef"
# show a message in the ScriptFTP window
IF(($A<=4) AND ($mytext=="abcdef"))
        PRINT ("Both conditions were true")
END IF

# Assign to $mylogicalvalue the result of
# two comparasion combined with the OR operator
$mylogicalvalue= (3 > $a) OR (4<2)

# As the first comparasion is true and the second
# is false, $mylogicalvalue should store "TRUE"
PRINT ($mylogicalvalue)

#Connect to the server and upload three files
OPENHOST ("ftp.myhost.com", "myuser", "123456")
$result1=PUTFILE ("C:\My Documents\notes.txt")
$result2=PUTFILE ("C:\theotherfile.txt")

```

```
$result3=PUTFILE ("C:\backup.zip")
CLOSEHOST ()

# Check if an error occurred:
# If result1 is different than "OK" or
# result2 is different than "OK" or
# result3 is different than "OK" show an error
IF(($result1!="OK") OR ($result2!="OK") OR ($result2!="OK"))
    PRINT ("An error was found uploading files")
ELSE
    PRINT ("Files uploaded correctly")
END IF
```

Text value operators

- Concatenate two values or variables.

```
$text1="this"
$text2="is"
$text3="ScriptFTP"

# Show "thisisScriptFTP" on the ScriptFTP window
$complete_text_without_spaces=$text1.$text2.$text3
PRINT($complete_text_without_spaces)

# Show "this is ScriptFTP" on the ScriptFTP window
$complete_text_with_spaces=$text1." ".$text2." ".$text3
PRINT($complete_text_with_spaces)

# Print "Hello World!" three times on the ScriptFTP window
PRINT("Hello Word!")
PRINT("Hello"." ". "Word!")
PRINT("He"."llo"." Wo"."rd!")
```

Commands for server connection

OPENHOST

Connect to an FTP server.

Syntax: OPENHOST(server,user,password)

- **server:** DNS name or IP address of the FTP server.
 - **user (optional):** user name for the login.
 - **password (optional):** login password.
-

Remarks:

- Use [SETPROTOCOL](#) before calling OPENHOST if you want to use **secure FTP**.
- If no user name and password are supplied ScriptFTP will try to login anonymously.
- If you do not want your FTP password to show up in clear text you can encrypt the entire script file. See [Encrypting script files](#).

Return value:

OPENHOST will return "OK" as soon as the login has been successful and the connection is established.

If the operation fails it will return an error code. You may check the return value in order to try again. See [Error handling](#) or the last example on this page.

See also:

[CLOSEHOST](#)
[SETPROTOCOL](#)
[SETPORT](#)
[SETTYPE](#)
[SETPASSIVE](#)

Examples:

```
# Connect to ftp.myhost.com as myuser and download all the files
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
GETFILE ("*.*")
CLOSEHOST
```

```
# Connect to ftp.funet.fi anonymously, download README
# and then upload it to ftp.myhost.com

# The file will be downloaded to the Windows temp directory
LOCALCHDIR ("C:\WINDOWS\TEMP")

OPENHOST ("ftp.funet.fi")
GETFILE ("README")
CLOSEHOST

OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

PUTFILE ("README")
CLOSEHOST

# Delete the downloaded README file
EXEC ("del README")
```

```
# This is a label marking a specific location in the script.
:start

# Connect to FTP server

$result=OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Check if OPENHOST has failed
```

```
IF ($result!="OK")

    PRINT ("Error. Cannot connect to FTP server.")
    # Wait 1 second
    SLEEP (1)
    # Retry
    GOTO :start

END IF

# Once ScriptFTP reaches this point
# it is connected to the FTP server

# Download notes.doc
GETFILE ("notes.doc")

# Close the connection
CLOSEHOST
```

CLOSEHOST

Close the current FTP connection.

Syntax: CLOSEHOST()

Remarks:

As this command does not accept any parameters the brackets are optional.

Return value:

CLOSEHOST will return "OK" if the connection has been closed successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error handling](#).

See also:

[OPENHOST](#)

Example:

```
# Connect to ftp.myhost.com as myuser and download all the files
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
GETFILE (*.*)
CLOSEHOST

# Connect to ftp.funet.fi anonymously, download README
# and then upload it to ftp.myhost.com

# The file will be downloaded to the Windows temp directory
LOCALCHDIR ("C:\WINDOWS\TEMP")

OPENHOST ("ftp.funet.fi")
GETFILE ("README")
CLOSEHOST

OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
```

```
PUTFILE ("README")
```

```
CLOSEHOST
```

```
# Delete the downloaded README file
```

```
EXEC ("del README")
```

SETPROTOCOL

Set the protocol used for the FTP connection. Use this command to enable secure FTP (FTPS).

Note: ScriptFTP currently supports FTP (common FTP) and FTPS (FTP over SSL). SFTP will be supported in future versions.

Syntax: SETPROTOCOL(protocol)

- **protocol:**

OFF	For standard FTP connections. This is the default value.
FTPS_EXPLICIT_ENCRYPT_DATA	Use this value if you want to encrypt file contents and login (user and password). ScriptFTP will connect to the FTP server using a standard FTP connection and will then send a specific command to enable SSL mode before logging in. Data connections used to transfer file contents will be encrypted.
FTPS_EXPLICIT	Use this value if you want to encrypt login (user and password) but don't care about file contents. ScriptFTP will connect to the FTP server using a standard FTP connection and will then send a specific command to enable SSL mode before logging in. Data connections used to transfer file contents will not be encrypted.
FTPS_IMPLICIT	Use this value if you want to encrypt file contents and login (user and password). Note that the FTP explicit mode

(FTPS_EXPLICIT and FTPS_EXPLICIT_ENCRYPT_DATA) is more widely adopted by FTP servers..

Implicit security mode will automatically establish an SSL connection as soon as ScriptFTP starts connecting to an FTP server. The port 990 will be used for this mode. Control and data connections will be encrypted.

Remarks:

Changes will take effect the next time [OPENHOST](#) is invoked in the script.

Standard FTP is used by default. There is no need to invoke SETPROTOCOL in your script if you do not want to use FTP over SSL (FTPS).

Depending on the selected protocol SETPROTOCOL will change the TCP port used for connecting to the FTP server. Use [SETPORT](#) **after** calling SETPROTOCOL if you want to set your own value.

Command History:

FTPS_EXPLICIT_ENCRYPT_DATA was added in ScriptFTP 2.1 build March 14th 2006

Return value:

This command always returns "OK".

See also:

[OPENHOST](#)

[SETPORT](#)

Example:

```
# Connect to ftp.myhost.com using FTPS
SETPROTOCOL (FTPS_EXPLICIT_ENCRYPT_DATA)
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
GETFILE ("*.*")
CLOSEHOST
```

```
# Connect to ftp.myhost.com using secure FTP
# download sales.xls and upload it to a local server

# The file is downloaded to the Windows temp directory
LOCALCHDIR ("C:\WINDOWS\TEMP")

# Use secure FTP
SETPROTOCOL (FTPS_IMPLICIT)
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

GETFILE ("sales.xls")
CLOSEHOST

# Go back to standard FTP
SETPROTOCOL (OFF)
OPENHOST ("192.168.1.53")
PUTFILE ("sales.xls")

CLOSEHOST

# Delete sales.xls
EXEC ("del sales.xls")
```

SETPORT

Set the server port used for connecting.

Syntax: SETPORT(port)

- **port:** port number.
-

Remarks:

Changes will take effect the next time [OPENHOST](#) is invoked in the script.

See also:

[OPENHOST](#)

[SETPROTOCOL](#)

Return value:

This command always returns "OK".

Examples:

```
# Connect to ftp.myhost.com using port 65002
SETPORT (65002)
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Download all files and disconnect
GETFILE ("*. *" )
CLOSEHOST

# Connect to ftp.myhost.com using
# implicit FTPS on the 3024 port
SETPROTOCOL (FTPS_IMPLICIT)
SETPORT (3024)
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Download all files and disconnect
GETFILE ("*. *" )
CLOSEHOST
```

Commands for file transfer

GETFILE

Download files from the FTP server.

Syntax: GETFILE(file,SUBDIRS)

- **file:** individual file name or wildcard expression for multiple files. The expression may include a remote directory, for example `"/htdocs/*.gif"`
 - **SUBDIRS (optional):** Download subdirectories.
-

Remarks:

This command will overwrite local files as needed.

Use [LOCALCHDIR](#) to set the current local directory before calling GETFILE. Files will be downloaded to that directory.

Use [SETPASSIVE](#) to disable passive transfer mode if you encounter firewall problems.

Use [SETTYPE](#) to change the data transfer type. Default is Binary (most common).

Use [SYNC](#) to download new or modified files only.

Return value:

GETFILE will return "OK" if every file has been downloaded successfully.

If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error Handling](#).

See also:

[SETPASSIVE](#)

[SETTYPE](#)

[PUTFILE](#)

[SYNC](#)

Example:

```
# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Change current local directory
LOCALCHDIR ("C:\dest_dir\excel_docs\")
```

```
# Download PriceList.xls
GETFILE("PriceList.xls")

# Change current local directory
LOCALCHDIR("C:\dest_dir\help/help/images\")

# Download all files from /images
GETFILE("/images/*.*)")

# Change current local directory
LOCALCHDIR("C:\dest_dir\")

# Download all files and directories
# from the remote directory /stuff
GETFILE("/stuff/*.*)",SUBDIRS)

# Close connection
CLOSEHOST
```

PUTFILE

Upload files to the FTP server.

Syntax: PUTFILE(file,SUBDIRS)

- **file:** individual file name or wildcard expression for multiple files. This parameter may include both absolute and relative paths, for example "C:\htdocs*.gif" or "htdocs*.gif"
 - **SUBDIRS (optional):** Upload subdirectories. If this parameter is supplied PUTFILE will upload entire directory trees.
-

Remarks:

This command will overwrite remote files as needed.

The wildcards supported are "*" and "?". If you are not familiar with the wildcard "?" see the example below.

The file parameter is case insensitive.

Use CHDIR to set the remote working directory before calling PUTFILE. Files will be uploaded to that directory.

Use SETPASSIVE to disable passive transfer mode if you encounter firewall problems.

Use SETTYPE to change the data transfer type. Default is Binary (most common).

Use SYNC instead of PUTFILE to upload new or modified files only.

Use a combination of GETLIST and FOREACH if you want to handle a set of files separately.

Return value:

PUTFILE will return "OK" if every file has been uploaded successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error Handling](#).

See also:

[SETPASSIVE](#)

[SETTYPE](#)

GETFILE

SYNC

Example:

```
# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Upload C:\htdocs\mydir\PriceList.xls
# to the remote directory /excel_docs
CHDIR ("/excel_docs")
PUTFILE ("C:\htdocs\mydir\PriceList.xls")

# Upload all files in C:\htdocs\help\images
# to the remote directory /help/images
CHDIR ("/help/images")
PUTFILE ("C:\htdocs\help\images\*.*)"

# Upload all files and directories in
# C:\htdocs\stuff to the remote directory /stuff
CHDIR ("/stuff")
PUTFILE ("C:\htdocs\stuff\*.*", SUBDIRS)

# Upload all zip files with a file name starting with "backup-"
# followed
# by two arbitrary characters and ending with "-old". For
# example,
# backup-54-old.zip
PUTFILE ("backup-??-old.zip")

CLOSEHOST
```

SYNC

Transfer new and modified files only. Optionally delete orphaned files.

Syntax: SYNC(local_dir, remote_dir, method,SUBDIRS,wildcard)

- **local_dir:** local directory to be synchronized. This directory must exist before calling SYNC.
- **remote_dir:** remote directory to be synchronized. This directory must exist before calling SYNC.
- **method:**

UPLOAD	Synchronize remote_dir from local_dir. All files in local_dir that do not exist in remote_dir will be uploaded. If a file exists in both locations it will be uploaded whenever the local_dir file is newer.
DOWNLOAD	Synchronize local_dir from remote_dir. All files in remote_dir that do not exist in local_dir will be downloaded. If a file exists in both locations it will be downloaded whenever the remote_dir file is newer.
UPLOAD_DELETE	Synchronize remote_dir from local_dir. All files in local_dir that do not exist in remote_dir will be uploaded. If a file exists in both locations it will be uploaded whenever the local_dir file is newer. Additionally, all remote_dir files that do not exist in local_dir (orphaned files) will be deleted.
DOWNLOAD_DELETE	Synchronize local_dir from remote_dir. All files in remote_dir that do not exist in local_dir will be downloaded. If a file exists in both locations it will be downloaded whenever the remote_dir file is newer. Additionally, all local_dir files that do not exist in remote_dir (orphaned files) will be deleted.

- **SUBDIRS** (optional): Use this parameter if you want ScriptFTP to synchronize subdirectories, too.

- **wildcard** (optional): Use this parameter to tell SYNC to synchronize only those files whose names match the wildcard. If this parameter is not used SYNC will synchronize all files.
-

Remarks:

The SYNC command always needs write permissions on the FTP site because it will create a temporary remote file for calculating the client-server time difference. This value is needed to find out whether a local file is newer than its remote counterpart (or vice versa). However, if you want to synchronize a local directory from a remote one (download) and do not have write privileges on the server you can set this time difference manually using the [SETCLOCKDIFF](#) command.

If DOWNLOAD_DELETE or UPLOAD_DELETE is used and SYNC is unable to delete certain orphaned files or directories it will merely display a warning. It will not stop and throw an error in this case.

SYNC will check whether local_dir and remote_dir already exist before starting the synchronization. If this check fails it will display an error and abort.

Command

compatibility:

ScriptFTP 2.1 Build Feb 2th 2006: The optional parameter "wildcard" is added.
ScriptFTP 2.2 Build April 4th 2006: The meaning of the third parameter has been changed. The UPLOAD and DOWNLOAD parameters do no longer perform any file deletion, use UPLOAD_DELETE and DOWNLOAD_DELETE instead.

Return value:

SYNC will return "OK" if all synchronization operations except from deleting orphaned files were successful.

If SYNC fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error Handling](#).

See also:

[GETFILE](#)

[PUTFILE](#)

[SETCLOCKDIFF](#)

Example:

```
# Connect to FTP server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Synchronize a local directory from an FTP site
SYNC ("C:\accounting", "/accounting", DOWNLOAD, SUBDIRS)

# Close connection
CLOSEHOST

# Connect to FTP server using FTP over SSL (secure)
SETPROTOCOL (FTPS_EXPLICIT)
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Synchronize a web site from a local directory
SYNC ("C:\local_website", "/www", UPLOAD_DELETE)

# Close connection
CLOSEHOST

# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Synchronize a single file only
SYNC ("C:\accounting", "/accounting", DOWNLOAD, "accounts.xls")

# Close connection
CLOSEHOST

# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Synchronize all Excel files from an FTP site
SYNC ("C:\accounting", "/accounting", DOWNLOAD, SUBDIRS, "*.xls")

# You may also omit the SUBDIRS parameter:
# SYNC ("C:\accounting", "/accounting", DOWNLOAD, "*.xls")

# Disconnect
CLOSEHOST
```


ADDEXCLUSION

Add files to the exclusion list. Use this command to have ScriptFTP ignore a set of files when transferring data.

Syntax: ADDEXCLUSION(list,filename,path)

- **list:** ScriptFTP maintains two exclusion lists: one for uploads and another one for downloads. Use this parameter to indicate which list you want to add the file to.
 - UPLOAD • Select the upload exclusion list.
 - •
 - DOWNLOAD • Select the download exclusion list.
 - **filename:** local or remote filename (depending on the list you have selected). Wildcards are also supported. **Do not include any path in this parameter**, only file names.
 - **path (optional):** local or remote path to the file you want to exclude. The path must be a full path, for example C:\Docs\Whatever\ or /remote_dir/whatever/. If you do not supply this parameter ScriptFTP will ignore all files matching the given file name regardless of their path.
-

Return value:

This command will return "OK" unless wrong parameters are supplied.

Remarks:

This command is capable of ignoring directories as well. Use "/" or "\" at the end of the file name to indicate that it is a directory. See the examples.

This command supports wildcards. You can use a wildcard expression in the file name parameter, for Example: "*.JPG" for ignoring all jpeg files. Note that file names on FTP servers are usually case sensitive; for example, *.jpg and *.JPG refer to different file extensions.

See also:[SYNC](#)[GETFILE](#)[PUTFILE](#)[CLEAREXCLUSION](#)**Example:**

```
# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Download
# Do not download tmp files, ignore the directory cgi-bin
ADEXCLUSION (DOWNLOAD, "*.tmp")
ADEXCLUSION (DOWNLOAD, "cgi-bin/")
LOCALCHDIR ("C:\dest_dir\")
GETFILE ("*.*", SUBDIRS)

# Upload
# Do not upload the backup directory, ignore all
# jpg files in C:\mydir\photos
LOCALCHDIR ("c:\mydir")
ADEXCLUSION (UPLOAD, "backup\")
ADEXCLUSION (UPLOAD, "*.jpg", "C:\mydir\photos")
PUTFILE ("*.*", SUBDIRS)

CLOSEHOST

OPENHOST ("ftp.myserver.com", "myuser", "123456")

# Download all files and directories in /d/server
# apart from all files (not directories) in /d/server/temp
ADEXCLUSION (DOWNLOAD, "*.*", "/d/server/temp")
GETFILE ("/d/server/*.*", SUBDIRS)
CLOSEHOST ()
```

CLEAREXCLUSION

Clear the upload and download exclusion lists

Syntax: CLEAREXCLUSION()

Return value:

This command always returns "OK".

Remarks:

As this command does not accept any parameters the brackets are optional.

See also:

[ADDEXCLUSION](#)

Example:

```
OPENHOST ("ftp.myserver.com", "myuser", "123456")

# Download all files and directories in /d/server
# apart from all files (not directories) in /d/server/temp
ADDEXCLUSION (DOWNLOAD, "*.*", "/d/server/temp")
GETFILE ("/d/server/*.*", SUBDIRS)
CLOSEHOST ()

# Clear the exclusion list
CLEAREXCLUSION

# Download all files and directories in /e/server
# apart from .tmp files
ADDEXCLUSION (DOWNLOAD, "*.tmp")
GETFILE ("/e/server/*.*", SUBDIRS)
CLOSEHOST ()
```

SETTYPE

Set the file transfer type.

Syntax: SETTYPE(type)

- type:

BINARY Default and most common transfer type. Files will be transferred without applying any transformation.

ASCII In text files Unix and Windows systems use different characters like line feed or carriage return to indicate new lines etc. Choose this transfer type if you want special characters contained in text files to be transformed automatically, thus keeping text files readable on both Unix and Windows systems.

EBCDIC Some IBM mainframes employ EBCDIC for representing text characters. Use this transfer type to automatically transform characters and keep text files readable.

Remarks:

Binary transfer type is used by default. If you do not use a transfer type other than binary there will be no need to invoke SETTYPE in your script.

See also:

[SETPASSIVE](#)

[GETFILE](#)

[PUTFILE](#)

[SYNC](#)

Return value:

This command always returns "OK".

Example:

```
# Connect to ftp.myhost.com as myuser and download all  
# txt files using the ASCII transfer type.
```

```
# Go back to BINARY transfer type and download all image files
```

```
LOCALCHDIR("C:\dest_dir")
```

```
OPENHOST("ftp.myhost.com","myuser","mypassword")
```

```
SETTYPE(ASCII)
```

```
GETFILE("textfiles/*.*)"
```

```
SETTYPE(BINARY)
```

```
GETFILE("images/*.*)"
```

```
CLOSEHOST
```

SETPASSIVE

Enable or disable passive transfer mode. Passive is used by default. Should you experience firewall problems please read the remarks section carefully. Switching the transfer mode and/or configuring your firewall may help.

Syntax: SETPASSIVE(mode)

mode:

ENABLED Default. Enable passive transfer mode.

DISABLED Disable passive transfer mode. This mode is called Active in the FTP standard.

Return value:

This command always returns "OK".

Remarks:

When connecting to an FTP server the client usually opens port 21 on the server where the server is listening and waiting for incoming connections. You may change your FTP server configuration such that it listens on a different port, however, port 21 is the standard. Once the connection has been established the client will authenticate to the server and then this connection is the one client and server will use to 'chat' with each other. For file transfers this connection will not be used, rather a new connection will be established for each file in order to transport the file's data. There are two methods for opening these new data channels: Active and Passive. The purpose of the SETPASSIVE command is to select the method that ScriptFTP will use. By default ScriptFTP uses passive mode.

- **Active Mode:**

In Active mode (also called non passive) the client starts listening on port N+1 and sends the FTP command PORT N+1 to the FTP server. The server will then connect back to this data port of the client using its own local data port, which is port 20. The file's data will then be transferred using this connection.

From the client's perspective the following communication channels need to be allowed in its own firewall in order to support active mode FTP:

- Allow the connections to the port 21 of the server address (Client initiates connection).
- Client's port > 1024 from the server address (Server connects to the client's data port to transfer a file).

From the server's perspective the following communication channels need to be allowed in its own firewall in order to support active mode FTP:

- FTP server's port 21 from anywhere (Client initiates connection)
- FTP server's port 21 to ports > 1024 (Server responds to client's control port)
- FTP server's port 20 to ports > 1024 (Server initiates data connection to client's data port)
- FTP server's port 20 from ports > 1024 (Client sends ACKs to server's data port)

In order to avoid the server having to initiate the connection to the client a different method for FTP connections was developed. This is known as Passive mode and it is the mode that ScriptFTP uses by default.

In Passive mode the FTP client initiates the connection to the server, thereby solving the problem that a firewall has to filter the incoming connection from the server to the client's data port.

- **Passive mode:**

The client will issue the PASV command whenever file data needs to be transferred. As a result the server will open a random unprivileged port (P > 1024) and send a PORT P command back to the client. The client will then initiate the connection to port number P on the server in order to transfer the file data.

From the client's perspective the following communication channels need to be allowed in its own firewall in order to support active mode FTP:

- Allow the connections to the port 21 of the server address (Client initiates connection).

- Allow the connections to ports > 1024 of the server address (Client connects to the server's data port to transfer a file).

From the server's perspective the following communication channels need to be opened in its own firewall in order to support passive mode FTP:

- FTP server's port 21 from anywhere (Client initiates connection)
- FTP server's port 21 to ports > 1024 (Server responds to client's control port)
- FTP server's ports > 1024 from anywhere (Client initiates data connection to random port specified by server)
- FTP server's ports > 1024 to remote ports > 1024 (Server sends ACKs (and data) to client's data port)

See also:

[GETFILE](#)

[PUTFILE](#)

[SYNC](#)

Example:

```
# Connect to ftp.myhost.com, download sales.xls
# using active mode and upload it
# to a local server using passive mode

# The file is downloaded to the Windows temp directory
LOCALCHDIR ("C:\WINDOWS\TEMP")

OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
# Use active transfer mode
SETPASSIVE (DISABLED)
GETFILE ("sales.xls")
CLOSEHOST

OPENHOST ("192.168.1.53")
# Go back to passive mode
SETPASSIVE (ENABLED)
```

```
PUTFILE ("sales.xls")  
CLOSEHOST  
  
# Delete sales.xls  
EXEC ("del sales.xls")
```

SETSPEED

Limit ScriptFTP's transfer speed

Syntax: SETSPEED(speed)

- **speed:** maximum speed allowed in KBytes/second.
-

Return value:

This command always returns "OK".

See also:

[GETFILE](#)

[PUTFILE](#)

[SYNC](#)

Example:

```
# Connect to ftp.myhost.com as myuser and download all files
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")
# Limit the maximum transfer speed
SETSPEED (20)
GETFILE ("*.*", SUBDIRS)
CLOSEHOST
```

SETCLOCKDIFF

Manually set the time difference between client and server.

Syntax: SETCLOCKDIFF(seconds)

- **seconds:** time difference in seconds between the FTP server and the PC where ScriptFTP is running. It is calculated via the formula: *Server time - PC time*.

Remarks:

In order to synchronize ([SYNC](#)) files ScriptFTP needs to know the time difference between the PC it is running on and the FTP server. By default the calculation of this time difference is performed automatically when you call [SYNC](#). However, if [SYNC](#) reports an error calculating this time difference you will have to use SETCLOCKDIFF in order to set it manually. Also see Transferring modified files only.

Return value:

This command always returns "OK".

Example:

```
# Connect to FTP server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Synchronize a local directory with this FTP site.
# The time difference needs to be set manually because
# we do not have write permissions on the FTP
# server.
SETCLOCKDIFF (3600)
SYNC ("C:\accounting", "/accounting", DOWNLOAD, SUBDIRS)

# Close connection
CLOSEHOST
```

SETUPLOADMODE

Specify how ScriptFTP uploads files.

Syntax: SETUPLOADMODE(mode)

TEMP This is the default mode. ScriptFTP will upload the files with a temporary filename (appending .part to the filename). As soon as the upload has completed ScriptFTP will rename the file to its original filename (removing the .part suffix)

DIRECT Use this mode to upload the files directly with their original filename.

Remarks:

Uploading files with DIRECT mode is not recommended. If ScriptFTP is uploading a file that already exists on the server it will be overwritten. If the transfer is interrupted the remote file will become corrupted. However, some FTP servers store the files being uploaded in a temporary directory until the upload is completed by the client. If this is true in your environment no file corruption will occur even if the transfer is interrupted and the mode has been set to DIRECT.

See also:

[SYNC](#)

[PUTFILE](#)

Example:

```
# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Change current remote directory
CHDIR ("/help/help/images")

# The FTP server does not allow
# renaming so we set the upload
# mode to DIRECT
SETUPLOADMODE (DIRECT)
```

```
# Upload all files in the directory images  
PUTFILE ("C:\htdocs\help\help\images\*.*)"
```

Commands for directory handling

LOCALCHDIR

Change the current local directory.

Syntax: LOCALCHDIR(path)

- **path:** absolute or relative local path, e.g. C:\Users\John\Docs\ or John\Docs
-

Remarks:

Use LOCALCHDIR to set the current local directory before calling [GETFILE](#). Files will be downloaded to that directory.

Return value:

LOCALCHDIR will return "OK" if ScriptFTP can access the remote directory. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error handling](#).

See also:

[CHDIR](#)

[LOCALCWDIR](#)

[LOCALRMDIR](#)

[LOCALMKDIR](#)

Examples:

```
# Connect to server
OPENHOST ("ftp.myftp.com")

# Change the current local directory
# All files will now be downloaded here
LOCALCHDIR ("C:\backup\web")

# Download files
GETFILE ("web/*.*", SUBDIRS)

# Go to the parent directory
LOCALCHDIR ("..")
```

```
# Download files  
GETFILE ("pricelist.xls")  
  
# Close connection  
CLOSEHOST
```

LOCALCWDIR

Get the current local directory.

Syntax: LOCALCWDIR()

Remarks:

As this command does not accept any parameters the brackets are optional. This command does not produce any output in the ScriptFTP window. It will always be silent.

Return value:

This command never reports an error. The return value always is the current local directory.

See also:

[LOCALRMDIR](#)

[LOCALMKDIR](#)

[CWDIR](#)

[CHDIR](#)

Example:

```
# Connect to server
OPENHOST("ftp.myhost.com","myuser","mypassword")

# Show the current local directory
$a=LOCALCWDIR()
PRINT($a)

# Change current local directory
LOCALCHDIR("C:/backup/docs/")

# This syntax is also valid
$a=LOCALCWDIR

# Should show C:/backup/docs
```

```
PRINT ($a)
```

```
# Close the connection
```

```
CLOSEHOST
```

LOCALMKDIR

Create a local directory.

Syntax: LOCALMKDIR(directory)

- **directory:** local directory to be created.

Return value:

LOCALMKDIR will return "OK" if the local directory has been created successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See Error handling.

See also:

[MKDIR](#)

[LOCALCWDIR](#)

[LOCALRMDIR](#)

[LOCALCHDIR](#)

Example:

```
# Create a local dir
LOCALMKDIR ("C:\foo")

# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Go to C:\dest_dir
LOCALCHDIR ("C:\dest_dir\")

# Create C:\dest_dir\images
LOCALMKDIR ("images")

# Download all images from the FTP site
GETFILE ("images/*.*.*)

CLOSEHOST
```

LOCALRMDIR

Remove a local directory.

Syntax: LOCALRMDIR(directory)

- **directory:** relative or absolute path to the local directory you want to delete
-

Remarks:

Relative path means that if the current directory is c:\ you only need to give the subdirectory, for example "windows". An absolute one means a complete path, for example "C:\windows\".

The directory to be removed must be empty. In order to delete a non-empty directory use the EXEC command to call the external "rd" command. For Example: EXEC("rd /s /q directory_name").

See also:

[RMDIR](#)

[LOCALMKDIR](#)

[LOCALCWDIR](#)

[LOCALCHDIR](#)

Return value:

If this command is executed successfully it will return "OK" . If it encounters an error it will return an error code.

Example:

```
#delete a local dir  
LOCALRMDIR ("foo")
```

CHDIR

Change the current remote directory.

Syntax: CHDIR(path)

- **path:** absolute or relative remote path, e.g. /www/htdocs/cgi-bin or htdocs/cgi-bin

Return value:

If this command is executed successfully it will return "OK" . If it encounters an error it will return an error code.

See also:

[CWDIR](#)

[MKDIR](#)

[RMDIR](#)

Example:

```
# Connect to ftp.myftp.com
OPENHOST ("ftp.myftp.com", "myuser", "mypassword")

# Change current remote directory to /help/images
CHDIR ("/help/images")

# Change current local dir to C:\dest_dir\help/images\
LOCALCHDIR ("C:\dest_dir\help/images\")

# Download all files from the current remote
# directory which is now /help/images
GETFILE ("*. *" )

CLOSEHOST
```

CWDIR

Get the current remote directory.

Syntax: CWDIR()

Remarks:

As this command does not accept any parameters the brackets are optional. This command will not produce any output in the ScriptFTP window. It will always be silent.

Return value:

The return value is the current remote directory. If ScriptFTP is not connected to an FTP server it will return nothing.

See also:

[CHDIR](#)

[MKDIR](#)

[RMDIR](#)

[LOCALCWDIR](#)

Example:

```
# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Create mydir
MKDIR ("mydir")

# Go to mydir
CHDIR ("mydir")

# Get the current directory
$a=CWDIR ()

# The print command should print mydir
PRINT ($a)
```

```
# Close the connection
```

```
CLOSEHOST
```

MKDIR

Create a remote directory.

Syntax: MKDIR(directory)

- **directory:** remote directory to be created.
-

Return value:

MKDIR will return "OK" as soon as the remote directory has been created successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error handling](#).

See also:

[LOCALMKDIR](#)

[CHDIR](#)

[CWDIR](#)

[RMDIR](#)

Example:

```
# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Create mydir
MKDIR ("mydir")

# Go to mydir
CHDIR ("mydir")

# Get the current directory
$a=CWDIR ()

# The print command should print mydir
PRINT ($a)

# Close the connection
CLOSEHOST
```


RMDIR

Delete a remote directory.

Syntax: RMDIR(directory)

- **directory:** remote directory to be deleted.
-

Remarks:

If the remote directory is not empty ScriptFTP will delete all files and subdirectories within that remote directory.

Return value:

RMDIR will return "OK" as soon as the remote directory has been removed successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error handling](#)

See also:

[LOCALRMDIR](#)

Example:

```
# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Remove the remote directory foo
RMDIR ("foo")

# Remove the remote directory temp in /www
RMDIR ("/www/temp")

# Close the connection
CLOSEHOST
```

Commands for file handling

RENAMEFILE

Rename remote files.

Syntax: RENAMEFILE(original_name,new_name)

- **original_name:** file name or wildcard expression describing multiple files.
 - **new_name:** the new file name or a wildcard expression describing multiple files.
-

Remarks:

For renaming multiple files only wildcard expressions of the form *.extension are allowed in original_name and new_name.

Return value:

RENAMEFILE will return "OK" once the files have been renamed successfully. If the operation fails it will return an error code. You may retrieve the return value and execute various operations depending on this value. See [Error handling](#).

Examples:

```
# Connect to server
OPENHOST ("ftp.myhost.com", "myuser", "mypassword")

# Rename foo.txt
RENAMEFILE ("foo.txt", "foo2.sav")

# Close connection
CLOSEHOST

# Connect to server
OPENHOST ("127.0.0.1", "myuser", "mypass")

# Change current remote dir to /files_to_rename
CHDIR ("/files_to_rename")

# Rename lenore.jpg to lenore.sav
```

```
RENAMEFILE ("lenore.jpg", "lenore.sav")

# Rename notes.txt to notes-(date).txt
$current_date=GETDATE (FORMAT3)
$new_file_name="notes-".$current_date.".txt"
RENAMEFILE ("notes.txt", $new_file_name)

# Shorter way:
RENAMEFILE ("notes.txt", "notes-".$current_date.".txt")

# Even shorter way:
RENAMEFILE ("notes.txt", "notes-".$GETDATE (FORMAT3) .".txt")

# Rename all jpg files to jpeg
RENAMEFILE ("*.jpg", "*.jpeg")

# Rename *2005.doc to *.doc.sav
RENAMEFILE ("*2005.doc", "*.doc.sav")

# Close connection
CLOSEHOST
```

DELETEFILE

Delete remote files.

Syntax: DELETEFILE(file)

- **file:** remote file to be deleted. Wildcard expressions and paths are allowed.
-

Remarks:

In order to delete an entire remote directory tree use [RMDIR](#)

Return value:

If this command has been executed successfully it will return "OK" . If it encounters an error it will return an error code.

See also:

[RMDIR](#)

Example:

```
# Connect to ftp.myserver.com
OPENHOST ("ftp.myserver.com", "myuser", "mypass")

# Delete the remote file notes.html
DELETEFILE ("notes.html")

# Delete the remote file lenore.sav
# under the directory foo
DELETEFILE ("foo/lenore.sav")

# Delete every .sav file
# under the current remote directory
DELETEFILE ("*.sav")

# Change current remote directory to
# /www
CHDIR ("/www")
```

```
# Delete every html file under  
# /www/images  
DELETEFILE ("images/*.html")  
  
# Close the connection  
CLOSEHOST
```

CHMOD

Change remote file permissions.

Syntax: CHMOD(mode,filename,SUBDIRS)

- **mode:** unix file mode to apply.
- **filename:** file name or wildcard expression.
- **SUBDIRS**(optional): Use this parameter for changing the permissions of remote files contained in subdirectories. Note that if you supply this parameter you will have to use a wildcard expression in the file name field.

Remarks:

If you want to apply CHMOD to a remote directory put the directory name in the file name field and append "/". For Example:

```
CHMOD(488,"/mydir/mysubdir/")
```

Note that this command is only available on Unix FTP servers.

Command History:

ScriptFTP 2.0.1 Build 14 Feb 2006: Added support for applying CHMOD to directories.
ScriptFTP 2.0.1 Build 21 Feb 2006: Added the optional parameter SUBDIR for applying CHMOD to files in subdirectories.

Return value:

CHMOD will return "OK" if the file permissions have been applied correctly. If it encounters an error it will return an error code.

Example:

```
# Connect to server  
OPENHOST("ftp.scriptftp.com","john","123456")
```

```
# Upload some files
PUTFILE ("*.php",SUBDIRS)

# Apply 667 to index.php in the current remote directory
CHMOD (667,"index.php")

# Apply 777 to all php files in the current remote directory
CHMOD (777,"*.php")

# Apply 777 to all php files in the /cgi-bin/ directory
CHMOD (777,"/cgi-bin/*.php")

# Apply 777 to all php files in the /cgi-bin/
# directory and subdirectories
CHMOD (777,"/cgi-bin/*.php",SUBDIRS)

# Apply CHMOD to mysubdir
CHMOD (488,"/mydir/mysubdir/")

# Close connection
CLOSEHOST
```

Handling local files

ScriptFTP does not provide any commands for copying, deleting or moving local files because these commands are already included in the operating system. The way to access them is using the EXEC command. For Example:

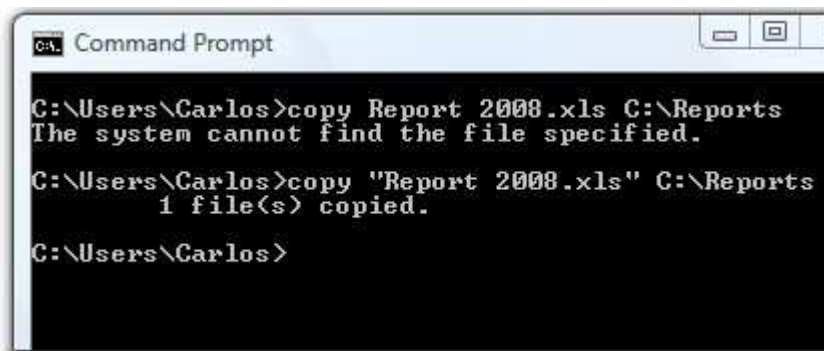
```
OPENHOST ("ftp.host.com", "myuser", "mypassword")

# Delete all temporary files from
# LocalWebFolder before synchronizing
EXEC ("del /F /Q C:\LocalWebFolder\*.tmp")

SYNC ("C:\LocalWebFolder", "/www", UPLOAD)
CLOSEHOST
```

In the above script, we use the EXEC command to call the external program "del". The result is the same as if you had opened a command line window and typed "del /Q C:\LocalWebFolder". For further information about using the Windows command line programs type "help" at the command line.

Note that some external commands such as copy, rename and del need the file name enclosed in double quotes when it contains spaces. For example, if we need to copy a file named *Report 2008.xls* to a directory whose path is C:\Reports:



```
ca. Command Prompt
C:\Users\Carlos>copy Report 2008.xls C:\Reports
The system cannot find the file specified.
C:\Users\Carlos>copy "Report 2008.xls" C:\Reports
1 file(s) copied.
C:\Users\Carlos>
```

The first attempt returned an error because copy were trying to find the files *Report* and *2008.xls* and they do not exist. In the second attempt we enclosed

the file name between quotes and it worked. Copy understood because of the quotes that both text strings were part of the same file name.

In this case, to execute the external command copy within ScriptFTP we also have to use double quotes to enclose the file name but the problem is that it seems that ScriptFTP also uses double quotes to separate command parameters. So, how can this command be written in a script without making a syntax error? Just use single quotes, ScriptFTP accepts both:

```
# Correct
EXEC('copy "Report 2008.xls" C:\Reports')

# Wrong. Copy will show an error because
# it tries to copy the files "Report" and "2008.xls"
# which do not exist.
EXEC("copy Report 2008.xls C:\Reports")

# Wrong. ScriptFTP will show a syntax error
# because of the EXEC command syntax, the quotes use
# is incorrect.
EXEC("copy "Report 2008.xls" C:\Reports")
```

Hint: Moving remote files

The FTP protocol is not capable of moving files on the FTP server. It was merely designed with file transfer in mind, but there is a simple trick that most FTP servers accept for moving a file: renaming. Let us look at an Example:

```
OPENHOST ("ftp.myserver.com", "carl", "123456")  
RENAMEFILE ("a.txt", "destination_folder/a.txt")  
CLOSEHOST
```

Note that this trick does not work for every FTP server. It usually works for Unix/Linux FTP servers only. If your server does not support it you can also do it downloading the file, removing it from its original remote directory and upload it again to its destination directory. It's a very slow way to move a remote file but at least works in every FTP server:

```
# Connect to FTP server  
OPENHOST ("ftp.myserver.com", "carl", "123456")  
  
# Change current local directory to c:\mytempdir  
LOCALCHDIR ("c:\mytempdir")  
  
# Download the file filetobemoved.zip from  
# the remote folder /origin_remote_dir  
GETFILE ("/origin_remote_dir/filetobemoved.zip")  
  
# Change the current remote directory to  
# /destination_remote_dir/  
CHDIR ("/destination_remote_dir/")  
  
# Upload the file filetobemoved.zip  
# to its destination  
PUTFILE ("filetobemoved.zip")  
  
# Delete the original remote file  
DELETEFILE ("/origin_remote_dir/filetobemoved.zip")  
  
# Close the connection  
CLOSEHOST
```

In the above script it is advisable to add some [error handling](#). We don't want to delete the file from the original location if the upload or any other step failed:

```
# Connect to FTP server
$result=OPENHOST("ftp.myserver.com","carl","123456")

# Stop the script execution if not connected
IF($result!="OK")
    STOP
END IF

# Change current local directory to c:\mytempdir
$result=LOCALCHDIR("c:\mytempdir")

# Stop the script execution if could not
# change the current local directory
IF($result!="OK")
    STOP
END IF

# Download the file filetobemoved.zip from
# the remote folder /origin_remote_dir
$result=GETFILE("/origin_remote_dir/filetobemoved.zip")

# Stop the script execution if could not
# download the file
IF($result!="OK")
    STOP
END IF

# Change the current remote directory to
# /destination_remote_dir/
$result=CHDIR("/destination_remote_dir/")

# Stop the script execution if could not
# change the current remote directory
IF($result!="OK")
    STOP
END IF
```

```
# Upload the file filetobemoved.zip
# to its destination directory
$result=PUTFILE("filetobemoved.zip")

# Stop the script execution if could not
# upload the file to the destination directory
IF($result!="OK")
    STOP
END IF

# Delete the remote file
$result=DELETEFILE("/origin_remote_dir/filetobemoved.zip")

# Stop the script execution if could not
# delete the remote file
IF($result!="OK")
    STOP
END IF

# Close the connection
CLOSEHOST
```

Script output commands

PRINT

Print text to the ScriptFTP screen.

Syntax: PRINT(text)

- **text:** text you want to be displayed.

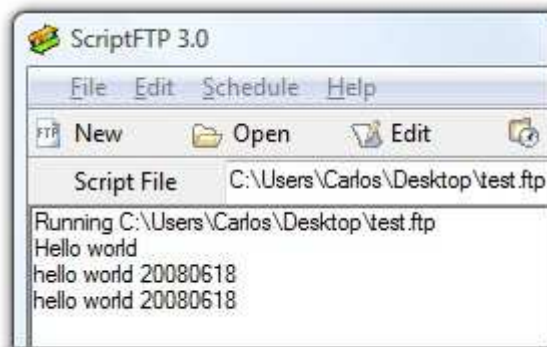
Return value:

This command always returns "OK".

Example:

```
# This script should display:  
# Hello world  
PRINT ("Hello world")  
  
# Store the current date in the $date variable  
$date=GETDATE (FORMAT3)  
  
# Build a text string concatenating  
# some words and the date variable  
$message="hello " ."world " .$date  
  
# Print the text  
PRINT ($message)  
  
# Shorter way:  
PRINT ("hello world " .GETDATE (FORMAT3))
```

The script output of the script above is the following:



SILENT

Make ScriptFTP less verbose

Syntax: Silent(mode)

- **mode:**
 - ON:** When silent is enabled ScriptFTP will only show errors, transferred files and the PRINT messages. All information other than that about the script run will be suppressed.
 - OFF:** Default. All messages are shown.
-

Return value:

This command always returns "OK".

See also:

[VERBOSE](#)

[LOGTO](#)

[PRINT](#)

Example:

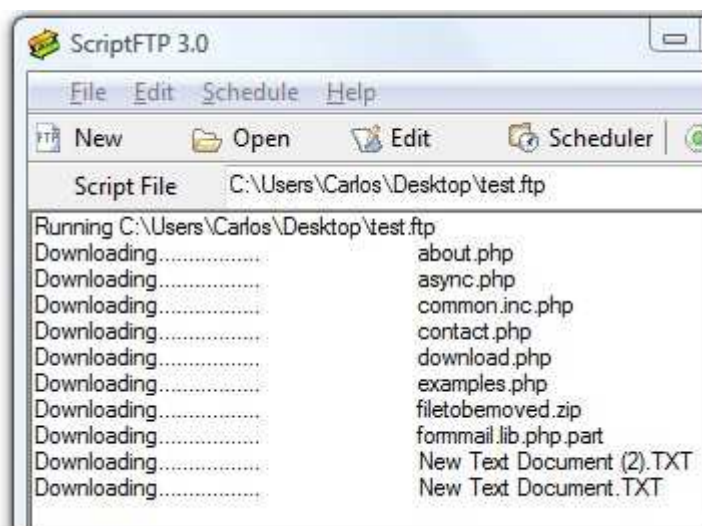
```
# Enable silent mode. Only transferred files and error will be
shown
SILENT (ON)

# Connect to FTP server
OPENHOST ("127.0.0.1", "carlos", "123456")

# Download some files
GETFILE ("*.*")

# Close the connection
CLOSEHOST
```

The script output of the above example is the following. Note that the commands are not shown:



Example:

```
# Display and log transferred files and errors only
SILENT (ON)

# Log script run to a text file
LOGTO ("C:\logs\TransferLog.txt")

# Connect to server
OPENHOST ("ftp.host.com", "myuser", "mypassword")

# Create the remote directory /sources
MKDIR ("/sources")

# Disable silent mode
SILENT (OFF)

# Change current remote directory
CHDIR ("/sources")

# Reenable silent mode
```

SILENT (ON)

Upload files

PUTFILE ("C:\PICS*.gif")

Close connection

CLOSEHOST

VERBOSE

Enable or disable displaying additional details about the script run.

Syntax: VERBOSE(mode)

- **mode:**
 - OFF:** This is the default. ScriptFTP will show the command calls, command messages and any errors encountered during the script run.
 - ON:** ScriptFTP will also show the dialog with the FTP server, DNS resolving issues and connection loss messages. Under certain circumstances it may also show internal messages.
-

Remarks:

Use this command to find out in detail what is happening during the script run. ScriptFTP will show the following information:

- The dialog between ScriptFTP and the FTP server.
- DNS resolving issues.
- Connection loss messages.
- Debug information about the status of ScriptFTP or the command currently executed.

This command is usually put at the beginning of a script file.

Return value:

This command always returns "OK".

See also:

[SILENT](#)

[LOGTO](#)

[PRINT](#)

Example:

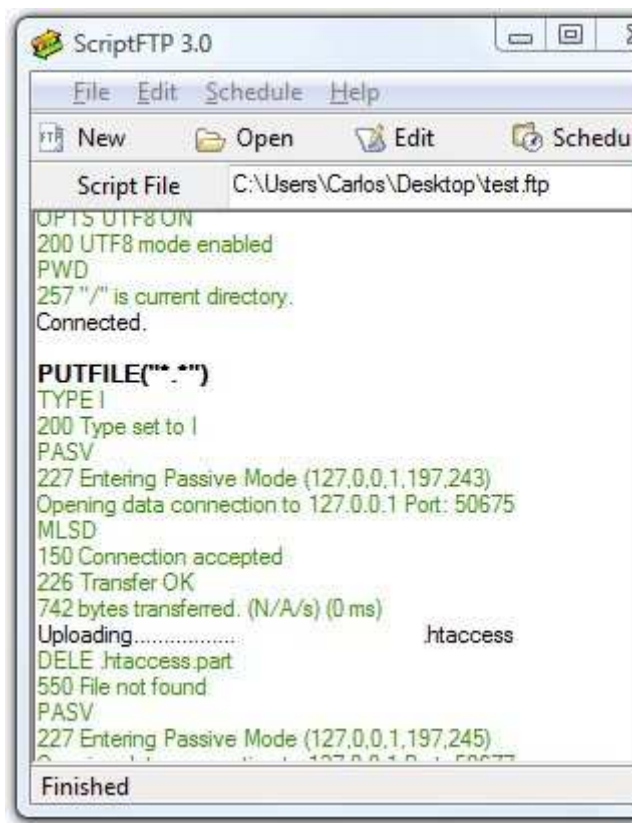
```
# Enable verbose mode
VERBOSE (ON)

# Connect to server
OPENHOST ("127.0.0.1", "myuser", "mypassword")

# Upload files
PUTFILE ("*.*)"

# Close connection
CLOSEHOST
```

The output of the script above is the following:



LOGTO

Log script messages to a text file.

Syntax: LOGTO(file,APPEND)

- **file:** name of the log file. A path may also be included.
 - **APPEND** (optional): append the script messages to the file instead of overwriting it.
-

Return value:

This command will return "OK" or else an error code if something went wrong opening the log file.

Remarks:

If the file exists it will be overwritten, use the APPEND parameter to change this behaviour.

See also:

[Logging ScriptFTP messages](#)

Example:

```
LOGTO ("C:\logs\TransferLog.txt",APPEND)
OPENHOST ("ftp.host.com", "myuser", "mypassword")
PUTFILE ("C:\PICS\*.jpg")
CLOSEHOST

# Get the current date
$date=GETDATE ()

# Concatenate "BACKUP", the current date and ".txt" to build the
log file name
$logfile="C:\BACKUP_LOGS\BACKUP-".$date.".txt"
```

```
# Start logging
LOGTO ($logfile)

OPENHOST ("ftp.host.com", "myuser", "mypassword")
PUTFILE ("C:\PICS\*.jpg")
CLOSEHOST
```

Miscellaneous commands

GETLIST

Get a file or directory listing of the local or remote site

Syntax: GETLIST(variable_name,list_type,filter)

- **variable_name:** variable GETLIST will save the resulting list to. Once GETLIST has finished you can use this variable in [FOREACH](#)
 - **list_type:** type of list to get.
 - LOCAL_FILES • Local file listing
 - LOCAL_DIRECTORIES • Local directory listing
 - REMOTE_FILES • Remote file listing
 - REMOTE_DIRECTORIES • Remote directory listing
 - **filter** (optional): if this parameter is used GETLIST will only include items in the listing which match the supplied wildcard.
-

Remarks:

Once a list is stored in a variable using this command you can perform a different action for each item in the list using the [FOREACH](#) loop.

As all ScriptFTP variables are of type string a file listing is also regarded as a string. Therefore GETLIST concatenates all item identifiers (file or directory names) using the character "|" as a separator before it saves them to the variable. You may check this output format by printing a file listing to the ScriptFTP window using [PRINT](#). This format is recognized by [FOREACH](#) loops.

Return value:

This command will return "OK" if the file listing has been received successfully. If your permissions do not suffice for a file listing or if you are not connected to the server GETLIST will return an error code. See [Error handling](#).

Command history:

This command was added in ScriptFTP 3.0 build July 28th 2008

See also:[FOREACH](#)**Examples:**

```
# Connect to server
OPENHOST ("ftp.myhost.com","joe","123456")

# Change the current local directory. All files
# will be downloaded here.
LOCALCHDIR ("C:\users\carlos\desktop\localftp")

# Get the remote file listing, store it in $list
GETLIST ($list,REMOTE_FILES,"*.txt")

# For each file in $list...
FOREACH $item IN $list
    # Download the file
    $result=GETFILE($item)
    # If the file has been downloaded successfully
    # delete the remote copy, else stop the script.
    IF ($result=="OK")
        DELETEFILE ($item)
    ELSE
        STOP
    END IF
END FOREACH

# Close the connection
CLOSEHOST

# Connect to server
OPENHOST ("127.0.0.1","carl","123456")

# Get the remote directory listing, store it in $list
GETLIST ($list,REMOTE_DIRECTORIES)

# For each directory in $list print the name
FOREACH $item IN $list
    PRINT ($item)
END FOREACH
```

```
# Get the local directory listing, store it in $list  
GETLIST ($list, LOCAL_DIRECTORIES)  
  
# For each directory in $list print the name  
FOREACH $item IN $list  
    PRINT ($item)  
END FOREACH  
  
# Close the connection  
CLOSEHOST
```

GETDATE

Get the current date and time

Syntax: GetDate(format)

- **format(optional):** The following formats are available:
 - FORMAT1
 - FORMAT2
 - FORMAT3
 - FORMAT4
 - YEAR
 - MONTH
 - DAY
 - HOUR
 - MIN
 - SEC
 - WEEKDAY
 - (hh: Hour, mm: Minute, DD: Day, MM: Month, YYYY: Year)
 - DD_MM_YYYY-hh_mm
 - YYYY_MM_DD-hh_mm
 - YYYYMMDD
 - hhmm
 - Current year in four digits format
 - Current month
 - Current day
 - Current hour
 - Current minute
 - Current second
 - Current day of week

Return value:

The requested date data.

Remarks:

The format parameter is optional, you may call GETDATE without any parameters. In this case it will default to FORMAT1.

Command History:

YEAR, MONTH, DAY, HOUR, MIN, SEC and WEEKDAY formats were added in ScriptFTP v2.2 Build 4th April 2006.

Examples:

```

# Print the current date in different formats:
PRINT (GETDATE (FORMAT1))
PRINT (GETDATE (FORMAT2))
PRINT (GETDATE (FORMAT3))
PRINT (GETDATE (FORMAT4))

# FORMAT1 is used by default:
PRINT (GETDATE)

# Get the current date
$date=GETDATE ()

# Concatenate "BACKUP", the current date and ".txt" to build the
log file name
$logfile="C:\BACKUP_LOGS\BACKUP-".$date.".txt"

# Start logging
LOGTO ($logfile)

OPENHOST ("ftp.host.com", "myuser", "mypassword")
PUTFILE ("C:\PICS\*.jpg")
CLOSEHOST

# Use GETDATE and CONCAT to build custom
# date formats
$mymonth=GETDATE (MONTH)
$myyear=GETDATE (YEAR)

# Concatenate text
$mydate=$mymonth."-".$myyear

PRINT ($mydate)

# Shorter way:
PRINT (GETDATE (MONTH) . "-" . GETDATE (YEAR) )

```

GETENV

Read a system environment variable.

Syntax: GETENV(variable)

- **variable:** name of the environment variable.
-

Examples:

```
# Show the path
p=GETENV("PATH")
PRINT(p)
```

```
# Log to C:\windows\temp\log-YYYYMMDD.txt
$logfile=GETENV("TEMP")."log-".GETDATE(FORMAT3)".txt"
LOGTO($logfile)

OPENHOST("ftp.scriptftp.com","john","123456")
PUTFILE("C:\ClientDocs\*.*",SUBDIRS)
CLOSEHOST
```

GETPARAM

Get a command line parameter.

Syntax: GETPARAM(param_number)

- **param_number:** position of the parameter in the command line sequence.
-

Remarks:

Use this command to pass your own command line parameters to your script file. The first parameter is the path of the ScriptFTP executable, the second one is the script file and all subsequent parameters are for your own use. Note that if you call ScriptFTP from a batch file or from the command line you will have to use double quotes in order to delimit the parameters that contain spaces. This may sound slightly complex but is pretty simple actually. For example:

```
C:\...\> ScriptFTP.exe script_file.exe
```

GETPARAM(5) will return *parameter* instead of *parameter five* and GETPARAM(6) will return *five*. In order to solve this issue you just have to insert double quotes:

```
C:\...\> ScriptFTP.exe script_file.exe paramter3 parameter4 "parameter five"
```

Now GETPARAM(5) will return *parameter five*.

Return value:

The parameter requested. If the parameter has not been supplied on the command line GETPARAM will return nothing.

See also:

[ScriptFTP in the command line](#)

Command

This command was added in ScriptFTP 2.1 build March 14th 2006

History:

Example:

```
$param1 = GETPARAM (1)
$param2 = GETPARAM (2)
$param3 = GETPARAM (3)
$param4 = GETPARAM (4)
$param5 = GETPARAM (5)

PRINT ($param1)
PRINT ($param2)
PRINT ($param3)
PRINT ($param4)
PRINT ($param5)

# Calling ScriptFTP with the following line:
# scriptftp_console.exe this_script.ftp "my own param 1"
other_param2 param3
#
# will produce the output:
# C:\Program Files\ScriptFTP\ScriptFTP_console.exe
# this_script.ftp
# my own param 1
# other_param2
# param3

# This script is used to upload a specified file
# to the FTP server. Use it this way:
# ScriptFTP.exe this_script.ftp the_file_to_upload

$param3 = GETPARAM (3)

PRINT ("Uploading file " . $param3)

OPENHOST ("127.0.0.1", "carl", "123456")

PUTFILE ($param3)

CLOSEHOST
```

RAWCOMMAND

Send a raw command to the FTP server.

Syntax: RAWCOMMAND(command)

- **command:** complete command including any parameters.

Remarks: Some FTP servers are capable of special or non-standard operations that are not supported directly by the set of commands ScriptFTP provides. RAWCOMMAND allows you to execute these operations by sending the given command to the FTP server. ScriptFTP will not carry out any further processing of the command you indicate. It will just pass it on to the server via the FTP control connection and then wait for a response.

Return value:

RAWCOMMAND will return the reply code sent by the FTP server.

Example:

```
# Adding an FTP user within an FTP session is  
# supported by the Apache FTP server. This  
# script adds the user carl and then lists the  
# current FTP users.  
  
OPENHOST("ftp.myhost.com","myuser","mypassword")  
  
# Add a FTP user  
RAWCOMMAND("SITE ADDUSER carl")  
  
# List users  
RAWCOMMAND("SITE LISTUSER")  
  
CLOSEHOST
```

EXEC

Run an external program or command.

Syntax: EXEC(command)

- **command:** external program or command including any parameters.
-

Remarks:

Some external commands require parameters to be enclosed in double quotes. If you merely fill in the double quotes ScriptFTP will display a syntax error, for example:

```
# This will produce a syntax error:  
EXEC("copy "C:\path with spaces\Report 2008.txt" C:\saved")
```

In order to resolve this problem you will have to use ' instead of ". For example:

```
EXEC('copy "C:\path with spaces\Report 2008.txt"  
C:\saved\notes.sav')
```

Return value:

This command will return the called program's exit code.

Example:

```
# Delete all doc files in the current local dir  
EXEC("del *.doc")  
  
# Call an external batch file  
EXEC("mybatch.bat")  
  
$result=EXEC("del *.mf")  
IF($result!=0)  
    IF($result==1)
```

```
    print("No *.mf files found. Errorcode: ".$result)
ELSE
    print("Del failed. Errorcode: ".$result)
    exit(1969)
END IF
ELSE
    print("deleting files was successful")
END IF
```

SLEEP

Wait for n seconds.

Syntax: SLEEP(n)

- **n:** number of seconds to wait.

Remarks:

If seconds is not a valid numeric value SLEEP will wait for 0 seconds.

Example:

```
# print "Hello world" every two seconds
while ("TRUE")
    print("hello world")
    sleep(2)
end while
```

STOP

Stop the script run

Syntax: STOP()

Remarks:

Brackets are optional.

Command history:

This command is available from ScriptFTP 3.0.

Example:

```
$result=OPENHOST ("ftp.myhost.com", "myuser", "mypass")

# If result is different than OK then stop the script execution
IF ($result!="OK")
    STOP
END IF

# Send all the files and subdirectories in C:\MyDir to the server
PUTFILE ("C:\MyDir\*.*", SUBDIRS)

# Transfer finished, close the connection
CLOSEHOST

$result=OPENHOST ("127.0.0.1", "carlos", "123456")

# If result is different than OK jump to :failed_connection
IF ($result!="OK")
    GOTO :failed_connection
END IF

# Change current local directory
```

```

$result=LOCALCHDIR("C:\destination_dir")
IF($result!="OK")
    GOTO :failed_change_local_dir
END IF

# Download all the files and subdirectories to C:\destination_dir
$result=GETFILE("*.*",SUBDIRS)

# If result is different than OK jump to :failed_transfer
IF($result!="OK")
    GOTO :failed_transfer
END IF

# Transfer finished, close the connection
CLOSEHOST

# Done. Stop the script execution.
STOP

# Go here when the connection failed three times
:failed_connection
PRINT("Cannot connect to the FTP server. Stopping script
execution.")
STOP

# Go here when got transfer errors
:failed_transfer
PRINT("Error downloading files.Stopping script execution.")
STOP

# Go here when got transfer errors
:failed_change_local_dir
PRINT("Cannot access C:\destination_dir. Stopping script
execution.")
STOP

```

EXIT

Close ScriptFTP.

Syntax: EXIT(exit_status)

- **exit_status** (optional): [exit status](#) of the ScriptFTP process.
-

Remarks:

If the parameter exit_status is not used the exit status will be 0.

See also:

[STOP](#)

Example:

```
# Connect to FTP server
$result=OPENHOST("127.0.0.1","carl","123456")

# If the returned value is different than OK
# means that the command failed
IF($result!="OK")
    PRINT("Cannot connect to FTP server. Aborting.")
    EXIT(1)
END IF

# Change current local directory
$result=LOCALCHDIR("C:\Users\Johnny Knoxville\Documents\")

# If the returned value is different than OK
# means that the command failed
IF($result!="OK")
    PRINT("Cannot change current local dictory. Aborting.")
    EXIT(2)
END IF
```

```
# Upload the doc files in the current local directory
$result=PUTFILE ("*.doc")

# If the returned value is different than OK
# means that the command failed
IF ($result!="OK")
    PRINT ("Error uploading files to FTP server. Aborting.")
    EXIT (3)
END IF

# Reached this points means that every
# previous command was successful.
PRINT ("Done!")

# Close connection
CLOSEHOST

# If no parameter is used in EXIT
# 0 will be used as the exit status.
EXIT
```

TEXTCUT

Returns a part of a text value.

Syntax: TEXTCUT(text,start_position,length)

- **text:** Source text.
 - **start_position:** The position of the first character in the text that is to be included in the returned text part.
 - **length:** Number of characters to extract from the starting position.
-

Return value:

This command returns the specified part of the source text. On error it returns an empty text value.

Remarks:

The start_position parameter is not zero-based. The first character is 1, the second 2 and so on.

Use -1 as the length parameter to get the rest of the text from the start position.

If length is higher than the rest of the available text from the start position the full remaining text will be returned.

See also:

[TEXTLENGTH](#)

Command History:

Added in ScriptFTP 3.1 build October 1th 2008

Example:

```
# Prints -this-  
$part=TEXTCUT("this is ScriptFTP",1,4)  
PRINT($part)
```

```
# Prints -La Mancha-
$part=TEXTCUT("In a village of La Mancha",17,9)
PRINT($part)

# Prints nothing, the second
# parameter (11) is wrong.
$part=TEXTCUT("whatever",11,8)
PRINT($part)

# Prints -123456789.txt-
$filename="prefix123456789.txt"
$filename2=TEXTCUT($filename,7,-1)
PRINT($filename2)

# Prints -123456789-
$text1="123456789"
$text2=TEXTCUT($text1,1,999)
PRINT($text2)
```

TEXTLENGTH

Returns the number of characters of a text value

Syntax: TEXTLENGTH(text)

- **text:** source text.
-

Return value:

The number of characters of the parameter text.

See also:

[TEXTCUT](#)

Command History:

Added in ScriptFTP 3.1 build October 1th 2008

Example:

```
# Prints 12
$count=TEXTLENGTH("whatever 123")
PRINT($count)

# Prints the file name without the extension
$filename="testing.txt"
$filename_without_extension=TEXTCUT($filename,1,TEXTLENGTH($filename)-4)
PRINT($filename_without_extension)
```

Advanced topics and rare features

This help topic documents some rare and not very commonly used ScriptFTP features. Some users may find them useful and eventually, if any become popular, they may get its own help topic.

ScriptFTP.exe command line options

The syntax of ScriptFTP.exe is the following:

```
ScriptFTP.exe <script_file_path> [custom parameter |  
/AUTOCLOSE | /HIDE | /TRAY] [custom parameter]....[custom  
parameter]
```

It's very similar to the syntax of scriptftp_console.exe shown [on this topic](#), but you can also use /TRAY, /HIDE or /AUTOCLOSE in the second parameter to change the way ScriptFTP starts or closes:

- **/AUTOCLOSE** Closes ScriptFTP automatically once the script finishes
- **/HIDE** Starts ScriptFTP as a background process. No window is shown.
- **/TRAY** Starts ScriptFTP as a tray icon.



These features were added in ScriptFTP 3.1

Special characters

Added in ScriptFTP 3.1. The command CHR() can be used to get a character from its [ASCII value](#). For example:

```
# It should print:
# abc
# def
$string = "abc".CHR(13)."def"
PRINT ($string)
```

Operations with dates

Added in ScriptFTP 3.1.

```
# Get the current date in the format AAAA_MM_DD-hh_mm_ss
$currentdate=GETDATE (FORMAT0)

# Then, you can subtract or add seconds to that date stored
# in the variable. Yesterday is:
$yesterday=$currentdate-(60*60*24)
# (3600*60*60) is the amount in seconds in a day.

# This should print the yesterday date:
PRINT ($yesterday)
```

Transferred files counter

Added in ScriptFTP 3.1.

```
SYNC ("C:\mylocaldir", "/myremotedir", UPLOAD)

IF (GETTRANSFEREDFILESCOUNT() > 0)
    PRINT ("Some files were transferred")
ELSE
    PRINT ("No files transferred")
END IF
```

Stop file logging

```
# This script logs only the download process  
# it used the STOPLOG command to stop any  
# further logging  
  
OPENHOST ("ftp.myhost.com", "myuser", "123456")  
  
LOGTO ("MyDownloadlog.txt")  
GETFILE ("*.*)"   
STOPLOG ()  
  
CLOSEHOST
```

License Agreement

The following terms and conditions must be read before using a ScriptFTP Software product. Unless you have a different license agreement signed by ScriptFTP Software you indicate your acceptance of this license agreement and the warranty conditions by making use of this ScriptFTP Software product.

Evaluation and Registration

This is not free software. You are hereby granted a license to use this software for evaluation purposes for a period of 30 days free of charge. If you continue to use this software after the 30-day evaluation period, a registration fee is required. When your payment is received a registration code will be sent to you by email.

ScriptFTP single user licenses are intended for individual use of this software and you are not allowed to install ScriptFTP on multiple workstations or servers if you purchased this license.

ScriptFTP site licenses grant you the right to install ScriptFTP on multiple computers that belong to the same institution. The number of installations is unlimited. You are also eligible for priority technical support.

ScriptFTP custom development licenses belong to a personalized license agreement signed by ScriptFTP Software. They permit you to use a customized and adapted version of ScriptFTP.

Unregistered use of the software after the 30-day evaluation period is in violation of U.S. and international copyright laws.

Disclaimer Warranty

This software and the accompanying files are sold "as is" and without warranties as to performance of merchantability or any other warranties whether expressed or implied.

No warranty of fitness for a particular purpose is offered. Good data processing procedures dictate that any program be thoroughly tested with non-critical data before relying on it. The user must assume the all risk when using the program. Any liability of the seller will be limited exclusively to product replacement or refund of purchase price.

Upgrades

Upgrades are provided for free for registered customers. License agreements of later versions supersede any prior agreements.

Distribution

Provided verification that you are distributing the Shareware Version you are hereby licensed to make as many copies of the Shareware version of this software and documentation as needed; distribute identical copies of the original Shareware version; and distribute the Shareware version of the software and documentation in its unmodified form via electronic means. There is no additional charge for any of the above.

You are specifically prohibited from charging, or requesting donations, for any such copies, however made; and from distributing the software and/or documentation with other products (commercial or otherwise) without prior written permission.