

FTP Scripting Manual

Table of Contents

Introduction	iv
1. Basic syntax	1
Commenting your script	2
Script commands	2
Predefined Status flags and constants	3
Strings, numbers, variables and lists	3
Conditional execution	4
Loops	5
List manipulation loops	6
Error checking	7
Adding execution delays	8
Forced exit from loops	8
Script termination	9
2. Working with string and numeric variables	10
Setting and updating variables	11
Printing variables	11
Incrementing and decrementing numeric variables	12
Converting strings to uppercase or lowercase	12
Modifying string contents	13
Comparing strings	15
3. File transfer commands	18
Connecting to remote hosts	19
Local and remote paths	21
Obtaining folder listings	22
Transferring files and folders	23
Dealing with duplicate files	24
Passive mode transfers	25
ASCII and binary transfer types	26
Non RFC959 compliant FTP servers	26
4. File system commands	28
Renaming files and folders	29
Deleting files and folders	29
Creating folders	30
Executing custom remote FTP commands	30
Executing local programs	31
Compressing and uncompressing local files and folders	31
Copying and moving local files and folders	33
5. Reading and writing local files	34
Opening and closing local files	35
Reading lines	36
Writing lines	36
6. Email notification	38
Creating email messages	39

Multi line messages and attachments	39
Sending email	40
7. Error reporting	42
Setting program exit codes	43
Writing out error files	43
Email notifications	43
8. Working with timestamps	45
Generating timestamps	46
Using timestamps	46
9. Accessing system information	48
Reading environmental variables	49
Getting the local computer name	49
10. Command line script parameters	51
Running a script from the command line	52
Specifying log files	52
Passing Environmental variables	52
Index	54

Introduction

FTP scripts enable the automation of file transfers and many other file processing activities. The scripting language is simple yet expressive and provides for error checking, conditional execution, looping, list handling, wildcard matching, and variable manipulation. Secure file transfers using both FTPS (SSL/TLS) and SFTP (SSH2) are supported in addition to regular FTP. Execution status can be communicated through exit codes, log files, error files, or even by sending a status email from within the script. Scripts may be executed from the command line, scheduled as one-time/recurring tasks, or called from other scripts.

This document contains the scripting language guide, a cookbook of frequently scripted operations, and sample FTP scripts. You may also contact our support team by filing a support ticket online at www.sysax.com for questions and help on writing FTP scripts.

1

Basic syntax

Commenting your script	2
Script commands	2
Predefined Status flags and constants	3
Strings, numbers, variables and lists	3
Conditional execution	4
Loops	5
List manipulation loops	6
Error checking	7
Adding execution delays	8
Forced exit from loops	8
Script termination	9

Commenting your script

Comments may be inserted into FTP scripts by preceding it with the "#" character. All text after "#" is ignored till the end of the current line.

Exhibit 1.1. Example of script comments

```
#this is a comment  
endscript; #this is comment
```

Script commands

All predefined script commands must be terminated with a semicolon. Parenthesis and commas between parameters are optional. Parenthesis are also optional for commands with no arguments.

Exhibit 1.2. Example of script commands

```
ftpconnect "127.0.0.1", 21, "anonymous", "anon@anon.com"; #command without  
parenthesis  
  
ftpconnect "127.0.0.1" 21 "anonymous" "anon@anon.com"; #command without  
parenthesis and commas  
  
ftpconnect("127.0.0.1", 21, "anonymous", "anon@anon.com"); #command with  
parenthesis and commas  
  
ftpdisconnect(); #command with no arguments with parenthesis  
  
ftpdisconnect; #command with no arguments without parenthesis
```

Predefined Status flags and constants

The **ftpresult** predefined status flag contains the result of commands that begin with the phrase *ftp*. It is set to the predefined constant **success** if the command completed successfully.

The **mailresult** predefined status flag contains the result of commands that begin with *mail*. It is set to the predefined constant **success** if the command completed successfully.

Other predefined constants are **true** and **false** that denote the corresponding conditions.

Strings, numbers, variables and lists

A string is any group of characters enclosed in quotes. Strings may also contain wildcards such as ***** and **?**. A number is any value between 0 and 2^{32} .

A variable is represented by a **~** character followed by alphanumeric or underscore characters. A variable may contain either string or numerical values and will be treated as a string or number depending on the context in which it is used. A variable is automatically created and initialized to an empty string at the point of first use.

Exhibit 1.3. Example of strings, numbers, and variables

```
"this is a string" #string
45 #number
~my_variable #variable
```

A list is represented by a **@** character followed by alphanumeric or underscore characters. Lists represent the contents of a folder and can contain zero or more items. Each item represents a specific file or folder. Lists can be set using the **ftpgetlist** command and each item in a list can be accessed using the **foreach** loop. Each list item is represented by a **\$** character followed by alphanumeric or underscore characters and contains the following properties:

```
.name          file or folder name string
.isfolder      flag set to false for file and true for folder
.size          size of the file in bytes
.modtime       modification time string in YYYYMMDDhhmmss format (Y=year,
M=month, D=day, h=hour, m=minute, s=second)
```

Exhibit 1.4. Example of lists

```
@my_list #list
$list_item #list item
$list_item.size #size property of list item
```

Conditional execution

A conditional statement consists of the keyword **if** followed by the test condition and a statement block. The statement block is mandatory. An **else** keyword may also be used to specify an alternate condition.

String comparison operators are:

```
eq      (equality)
ne      (inequality)
lt      (less than)
le      (less than or equal to)
gt      (greater than)
ge      (greater than or equal to)
```

Numeric comparison operators are:

```
==      (equality)
!=      (inequality)
<       (less than)
<=      (less than or equal to)
>       (greater than)
>=      (greater than or equal to)
```

Arguments are automatically treated as strings or numbers based on the comparison operator that is used. Strings may also contain wildcards such as `*`.

Exhibit 1.5. Syntax of conditional execution statement

```
if <test condition> begin
end

if <test condition> begin
end else begin
end
```

Exhibit 1.6. Example of conditional execution statement

```
if ~numvar < 5 begin
end

if ~strvar eq "*.txt" begin
end else begin
end
```

Loops

A loop statement consists of the keyword **loop**, followed by an integer number or a variable, followed by a statement block. A statement block begins with the **begin** keyword and ends with the **end** keyword. The statement block is mandatory.

Exhibit 1.7. Syntax of loop statement

```
loop <integer number> begin  
end
```

Exhibit 1.8. Example of loop statement

```
loop 25 begin  
end  
  
loop ~my_variable begin  
end
```

List manipulation loops

A list based loop statement consists of the keyword **foreach** followed by a list item name, the keyword **in**, the list name, and a statement block. The statement block is mandatory.

Exhibit 1.9. Syntax of list manipulation loop statement

```
foreach <list item name> in <list name> begin  
end
```

The list is filled using the **ftpgetlist** command and the foreach loop is used to access each file or folder item on the list.

Exhibit 1.10. Example of list manipulation loop statement

```
foreach $list_item in @my_list begin
    print "item name is ", $list_item.name, " and size is ",
    $list_item.size;
end
```

Error checking

The **ftpresult**, **mailresult**, and **fileresult** predefined status flags can be used to determine the result of the last executed command. The flags will be set to **success** if the command completed successfully. Errors may be reported by setting a program exit code, printing an error message, writing out an error file, or even sending out an email message.

Exhibit 1.11. Examples of error checking

```
ftpconnect "127.0.0.1", 21, "anon", "anon@anon.com";
if ftpresult eq success begin
end else begin
end

mailcreate "My name", "me@mydomain.com", "Test mail", "This is a test
mail";
if mailresult eq success begin
end else begin
end

fileopen read, "myfile.txt";
if fileresult eq success begin
end else begin
end
```

Adding execution delays

Script execution can be temporarily paused using the **waitsecs** command. The time to wait is specified in seconds.

Exhibit 1.12. Syntax of execution delay command

```
waitsecs <number of seconds>;
```

Exhibit 1.13. Example of execution delay command

```
waitsecs 30; #pause script execution for 30 seconds
```

Forced exit from loops

The **exitloop** command can be used to immediately exit from a loop. If this command is called from nested loops, the innermost loop is exited.

Exhibit 1.14. Syntax of loop exit command

```
exitloop;
```

Exhibit 1.15. Example of loop exit command

```
loop 100 begin
    ftpconnect "ftp.ftpsite.com", 21, "ftp", "ftp@site.com";
    if success eq ftpresult begin
        exitloop; #exit loop immediately if connected
    end
end
```

Script termination

A script will stop execution after the last command in the file is executed. The script can be terminated at any other execution point by calling the **endscript** command.

Exhibit 1.16. Example of script termination command

```
if ftpresult ne success begin
    endscript;
end
```

2

Working with string and numeric variables

Setting and updating variables	11
Printing variables	11
Incrementing and decrementing numeric variables	12
Converting strings to uppercase or lowercase	12
Modifying string contents	13
Comparing strings	15

Setting and updating variables

A variable is automatically created and set to an empty string when first used. The **setvar** command or the **strprint** command can be used to set or update the values stored in a variable. A variable holds both numbers and strings and can be treated as a number or string based on the context.

Exhibit 2.1. Syntax of commands for setting variables

```
setvar <variable>, <string or number>;  
  
strprint <variable>, <sequence of comma separated strings, numbers, and  
variables>;
```

Exhibit 2.2. Example of commands for setting variables

```
setvar ~my_number, 5;  
  
setvar ~my_string, "this is a string";  
  
strprint ~my_value, "the value is ", 5, " bytes";
```

Printing variables

The **print** command is used to print variables, strings, and numbers to the log file.

Exhibit 2.3. Syntax of commands for printing variables

```
print <variable>, <sequence of comma separated strings, numbers, and  
variables>;
```

Exhibit 2.4. Example of commands for printing variables

```
print ~my_value, "the folder contains ", 17, " files with a combined size  
of ", ~my_size, " Megabytes";
```

Incrementing and decrementing numeric variables

The **increment** command and **decrement** commands can be used to increment or decrement a numeric variable by 1.

The syntax is

Exhibit 2.5. table title

```
increment <variable>;  
decrement <variable>;
```

Some examples are

Exhibit 2.6. table title

```
setvar ~my_num, 4;  
increment ~my_num; #value of ~my_num is now 5  
decrement ~my_num; #value of ~my_num is now 4
```

Converting strings to uppercase or lowercase

The **strupper** and **strlower** commands are used to string variables into upper or lower case.

The syntax is

Exhibit 2.7. table title

```
strupper <variable>;  
strlower <variable>;
```

Some examples are

Exhibit 2.8. table title

```
setvar ~my_var, "NaMe.Txt";  
strupper ~my_var; #convert to uppercase  
strlower ~my_var; # convert to lowercase
```

Modifying string contents

The **strslice** command can be used to extract part of a string. The **left** and **right** keywords are used to specify the start direction for the extracted string slice. The offset specifies the number of characters to skip from the start direction. The character count specifies the number of characters to extract. A value of 0 for the character count extracts the entire remaining portion of the string. The extracted string replaces the original string contained in the variable.

The syntax is

Exhibit 2.9. table title

```
strslice <variable>, <keyword: left, right>, <offset from start  
direction>, <character count>;
```

Some examples are

Exhibit 2.10. table title

```
setvar ~my_var, "my_string_value";

strslice ~my_var, left, 4, 0; #start from left, skip 4 chars and extract
  rest of the string. ~my_var contains "tring_value"

strslice ~my_var, left, 0, 4; #start from left, extract 4 chars. ~my_var
  contains "my_s"

strslice ~my_var, right, 4, 0; #start from right, skip 4 chars and extract
  rest of the string. ~my_var contains "my_string_v"

strslice ~my_var, right, 0, 5; #start from right, extract 5 chars. ~my_var
  contains "value"
```

The **regexreplace** command replaces parts of a string with another string based on a regular expression pattern. This is similar to the substitution operation in Perl. The regex pattern specifies the regular expression pattern to search for in the original string. The replacement text specifies the string to substitute when a match is found. The match options specify other conditions that affect a match. The available match options are:

```
i    perform a case insensitive match
g    match all occurrences of a pattern
```

Note that for the regular expression pattern string, a `\` character must be preceded by another `\`.

The syntax is

Exhibit 2.11. table title

```
regexreplace <variable>, <regex pattern string>, <replacement text
  string>, <match options>;
```

Some examples are

Exhibit 2.12. table title

```
setvar ~my_var, "xyz_file_050502out.txt";

regexreplace ~my_var, "[0-9]+", "", "ig"; #remove all numbers in string.
~my_var contains "xyz_file_out.txt"

regexreplace ~my_var, "[0-9]+", "00000", "ig"; #replace numbers in string
with 00000. ~my_var contains "xyz_file_00000out.txt"
```

The replacement text can also contain backreferences. Backreferences are specified using \$1 for the subexpression matched in the first parenthesis, \$2 for the subexpression matched in the second parenthesis and so on.

Some examples are

Exhibit 2.13. table title

```
setvar ~my_var, "dat_file.txt";

regexreplace ~my_var, "([\_a-zA-Z0-9]+\)\.\.[\_a-zA-Z0-9]+", "$1.log", "ig";
#rename extension from txt to log. ~my_var contains "dat_file.log";

regexreplace ~my_var, "([\_0-9a-zA-Z]+)[.][\_0-9a-zA-Z]+", "$1_050402.txt",
"ig"; #append a timestamp. ~my_var contains "dat_file_050402.txt"
```

Comparing strings

Strings can be directly compared using the string comparison operators. Additionally, strings can also be matched using wildcards such as *, and ?. This is known as wildcard matching or pattern matching. * matches 0 or more characters while ? matches exactly one character.

Some examples are

Exhibit 2.14. table title

```
if "index.txt" eq ~my_var begin
end

if "*.txt" eq ~my_var begin
end
```

The **ftpwildcardmatch** command can be used to perform wildcard matching. Additionally, this command allows case insensitive pattern matching when **i** is passed as a match option. If the strings match, the predefined flag **ftpresult** is set to the predefined constant **success**.

The syntax is

Exhibit 2.15. table title

```
ftpwildcardmatch <string to match>, <wildcard pattern string>, <match
options>;
```

Some examples are

Exhibit 2.16. table title

```
ftpwildcardmatch ~my_var, "*.txt", "i"; #case insensitive wildcard match
if success eq ftpresult begin
end
```

The **ftpregexmatch** command can be used to perform matching using regular expressions similar to that used in Perl. This command allows case insensitive regular expression matching when **i** is passed as a match option. If the strings match, the predefined flag **ftpresult** is set to the predefined constant **success**. Note that for the regular expression pattern string, a **** character must be preceded by another ****.

The syntax is

Exhibit 2.17. table title

```
ftpregexmatch <string to match>, <regex pattern string>, <match options>;
```

Some examples are

Exhibit 2.18. table title

```
ftpregexmatch ~my_var, "[a-zA-Z0-9]+\.\log", ""; #case sensitive regex  
match  
if success eq ftpresult begin  
  
end
```

3

File transfer commands

Connecting to remote hosts	19
Local and remote paths	21
Obtaining folder listings	22
Transferring files and folders	23
Dealing with duplicate files	24
Passive mode transfers	25
ASCII and binary transfer types	26
Non RFC959 compliant FTP servers	26

Connecting to remote hosts

Connections to a remote FTP server are established using the **ftpconnect** command. The IP address of the remote server must be specified as the address string. The port number indicates the port where FTP connections are accepted by the remote server. The default port number for regular FTP connections is 21. The username and password strings are required to log into a specific user account. If the FTP server allows anonymous access, the username should be *ftp* and the password should be the user's email address. Some legacy FTP servers may also require an optional account string. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 3.1. table title

```
ftpconnect <address string>, <port number>, <username>, <password> [,
optional: <account>];
```

Secure connections using the SSL protocol (also known as FTPS) are established using the **ftpconnectssl** command. The **ftpconnectsslc** command can be used if only the control connection needs to be encrypted using SSL.

The syntax is

Exhibit 3.2. table title

```
ftpconnectssl <address string>, <port number>, <username>, <password> [,
optional: <account>];

ftpconnectsslc <address string>, <port number>, <username>, <password> [,
optional: <account>];
```

Client side SSL certificates may be used for authentication of a client to the remote server. The **certload** command enables a client side certificate to be loaded. The certificate must be loaded before establishing a connection using **ftpconnectssl** and must be in the *pem* format.

The syntax is

Exhibit 3.3. table title

```
certload <certificate file>, <private key file> [, optional:  
<passphrase>];
```

Secure connections using the SSH protocol (also known as SFTP) are established using the **ftpconnectssh** command. The default port number for ssh connections is 22.

The syntax is

Exhibit 3.4. table title

```
ftpconnectssh <address string>, <port number>, <username>, <password> [,  
optional: <account>];
```

Public key authentication for SSH connections allows clients to use a private key for authentication instead of a password. The **pkeyload** command enables a private key to be loaded to perform SSH public key authentication. The key must be loaded before establishing a connection using **ftpconnectssh** and must be in the *pem* format.

The syntax is

Exhibit 3.5. table title

```
pkeyload <private key file> [, optional: <passphrase>];
```

If it is necessary to connect using a http proxy server, the **proxyhttp** command should be called before establishing a connection.

The syntax is

Exhibit 3.6. table title

```
proxyhttp <proxy address>, <proxy port> [, optional: <proxy username>,  
<proxy password>];
```

The **ftpdconnect** command is used to close a connection that was established using any of the above commands.

The syntax is

Exhibit 3.7. table title

```
ftpdisconnect;
```

Some examples are

Exhibit 3.8. table title

```
ftpconnect "ftp.mysite.com", 21, "username", "password"; #connect to a
non-secure ftp server
if success eq ftpresult begin

end

certload "c:\\certs\\mycert.pem", "c:\\keys\\myprivkey.pem", "keypass";
ftpconnectssl "ftp.securesite.com", 21, "user", "pass"; # connect to an
SSL (FTPS) based secure server using client side certificate

pkeyload "c:\\keys\\privkey.pem", "keypass";
ftpconnectssh "ftp.securesite.com", 22, "", ""; #connect to an SSH (SFTP)
based secure server using public key auth

ftpdisconnect; #disconnect from the remote ftp server
```

Local and remote paths

The **ftpsetpath** command is used to set the current working path for the local and remote systems. The predefined keywords **local** or **remote** are used to specify either the local or remote system. The path string contains the new path to be set. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 3.9. table title

```
ftpsetpath <keywords: local, remote>, <path string>;
```

Some examples are

Exhibit 3.10. table title

```
ftpsetpath local, "c:\\ftptemp"; #set the current local working path  
ftpsetpath remote, "/home/ftpin"; "set the current remote working path
```

Obtaining folder listings

The contents of a folder can be listed and stored into a user specified list name. Each item in the list can then be individually accessed. The **ftpgetlist** command can be used to obtain a listing of the current working path. The predefined keywords **local** or **remote** are used to specify either the local or remote system. A user specified list name is used to store the listing result. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 3.11. table title

```
ftpgetlist <keywords: local, remote>, <list name>;
```

Some examples are

Exhibit 3.12. table title

```
ftpgetlist local, @local_list; #get the listing of the current local  
working path  
ftpgetlist remote, @remote_list; #get the listing of the current remote  
working path
```

Transferring files and folders

The **ftpdownload** command can be used to download either an individual file or an entire folder tree. The files and folders are downloaded to the current local working path that was set using the **ftpsetpath** command. The predefined keywords **file** and **folder** are used to specify either a file or a folder. An optional local name string can be provided to save the downloaded item to a new name. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 3.13. table title

```
ftpdownload <keywords: file, folder>, <remote name string> [, optional:  
<local name string>];
```

Some examples are

Exhibit 3.14. table title

```
ftpdownload file, "text.dat"; #download file  
ftpdownload file, "text.dat", "text_0503.dat"; #download file and save as  
text_0503.dat  
ftpdownload folder, "www"; #download entire folder tree
```

The **ftpupload** command can be used to upload either an individual file or an entire folder tree. The files and folders are uploaded to the current remote working path that was set using the **ftpsetpath** command. The predefined keywords **file** and **folder** are used to specify either a file or a folder. An optional remote name string can be provided to save the uploaded item to a new name. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 3.15. table title

```
ftpupload <keywords: file, folder>, <local name string> [, optional:  
<remote name string>];
```

Some examples are

Exhibit 3.16. table title

```
ftpupload file, "out_text.dat"; #upload file

ftpupload file, "out_text.dat", "out_text_0503.dat"; #upload file, saving
it as out_text_0503.dat

ftpupload folder, "www"; #upload entire folder tree
```

Dealing with duplicate files

During a download or an upload, if a file with the same name already exists in the destination path, the **setduperules** command can be used to specify a set of rules to compare the files by size or date and then choose to skip, overwrite, resume, or rename the file. The predefined keywords **bysize** and **bydate** are used to compare the files either by size or by date. The predefined keywords **skip**, **overwrite**, **resume**, and **rename** are used to specify the type of action to perform. An action is specified for each of the possible three cases - smaller/older, same size/date, and larger/newer. If the rename option is chosen for any of the three cases, an extension string to be used to rename the file can be optionally provided. The default rename extension is *.fbak*.

The syntax is

Exhibit 3.17. table title

```
setduperules <keywords: bysize, bydate>, <action if smaller/older>,
<action if same size/date>, <action if larger/newer> [, optional: <rename
ext string>];
```

Some examples are

Exhibit 3.18. table title

```
setduperules bysize, resume, skip, skip; #resume if smaller, skip if same
size or larger

setduperules bydate, overwrite, skip, skip; #overwrite if older, skip if
same date or newer

setduperules bysize, rename, skip, rename, ".0503"; #skip if same size,
rename with ext ".0503" if size is different
```

Passive mode transfers

Files may be transferred using server initiated (PORT) or client initiated (PASV) data connections. The **enablepasv** command enables PASV data connections while the **disablepasv** command enables PORT data connections. These commands do not apply to SSH based SFTP file transfers.

The syntax is

Exhibit 3.19. table title

```
enablepasv;
disablepasv;
```

Some examples are

Exhibit 3.20. table title

```
enablepasv; #enable PASV data connections
disablepasv; #enable PORT data connections
```

ASCII and binary transfer types

The **settransfertype** command is used to set the appropriate transfer type for the file being transferred. The *ASCII* transfer type is used to transfer text while the *binary* transfer type is used to transfer images. The *auto* transfer type tries to detect the file type based on its extension and performs either ASCII or binary transfer. The predefined keywords **ascii**, **binary**, and **auto** are used to select the desired transfer type.

The syntax is

Exhibit 3.21. table title

```
settransfertype <keywords: ascii, binary, auto>;
```

Some examples are

Exhibit 3.22. table title

```
settransfertype binary; #set transfer type to binary  
settransfertype ascii; #set transfer type to ascii
```

Non RFC959 compliant FTP servers

Some legacy FTP servers produce folder content listings in non-standard formats. In these cases, the **setnlst** command can be used to provide a name-only listing. When a listing is obtained in this manner, only the *.name* property of each list item is available for use in a foreach loop. Name-only listings may be turned off using the **unsetnlst** command.

The syntax is

Exhibit 3.23. table title

```
setnlst;  
unsetnlst;
```

Some examples are

Exhibit 3.24. table title

```
setnlst; #enable name-only listing  
unsetnlst; #disable name-only listing
```

4

File system commands

Renaming files and folders	29
Deleting files and folders	29
Creating folders	30
Executing custom remote FTP commands	30
Executing local programs	31
Compressing and uncompressing local files and folders	31
Copying and moving local files and folders	33

Renaming files and folders

Files and folders can be renamed using the **ftprename** command. The predefined keywords **local** and **remote** are used to apply this command on either the local or remote computer. Both the original name string and the new name string should be specified. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.1. table title

```
ftprename <keywords: local, remote>, <original name string>, <new name string>;
```

Some examples are

Exhibit 4.2. table title

```
ftprename local, "file_1.txt", "file_1_0305.txt"; #rename local file
ftprename remote, "file.log", "file_0305.log"; #rename remote file
```

Deleting files and folders

The **ftpdelete** command is used to delete files and folders. The predefined keywords **local** and **remote** are used to apply this command on either the local or remote computer. The predefined keywords **file** and **folder** are used to specify whether the item is a file or folder. Folders will be recursively deleted. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.3. table title

```
ftpdelete <keywords: local, remote>, <keywords: file, folder>, <name string>;
```

Some examples are

Exhibit 4.4. table title

```
ftpdelete local, file, "file_1.txt"; #delete local file  
ftpdelete remote, folder, "tmp_folder"; #delete remote folder and all  
sub-folders recursively
```

Creating folders

The **ftpmakefolder** command is used to create a new folder. The predefined keywords **local** and **remote** are used to apply this command on either the local or remote computer. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.5. table title

```
ftpmakefolder <keywords: local, remote>, <name string>;
```

Some examples are

Exhibit 4.6. table title

```
ftpmakefolder local, "tempfiles"; #create local folder  
ftpmakefolder remote, "tmp_folder"; #create remote folder
```

Executing custom remote FTP commands

The **ftpcustomcmd** command is used to execute a literal RFC959 based FTP command string on the remote server when connected to a non secure or SSL (FTPS) based secure FTP server. When connected to an SSH (SFTP) based secure server, a shell command string can be executed if permitted by the server. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.7. table title

```
ftpcustomcmd <command string>;
```

Some examples are

Exhibit 4.8. table title

```
ftpcustomcmd "SITE CHMOD 444 myfile.txt"; #run a site specific FTP command  
string
```

Executing local programs

The **ftprunprogram** command runs a program on the local computer. The path to the program executable and the parameters to be passed to the program should be specified. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.9. table title

```
ftprunprogram <program path string>, <parameter string>;
```

Some examples are

Exhibit 4.10. table title

```
ftprunprogram "c:\\Program Files\\notepad.exe", "C:\\myfile.txt"; #open a  
file in notepad
```

Compressing and uncompressing local files and folders

Files and folders can be compressed using the **ftpcompress** command. The name string specifies the file or folder to compress. If compression is successful, the output name variable contains the name of the compressed file and the **ftpresult** predefined status flag is set to the predefined constant **success**.

The syntax is

Exhibit 4.11. table title

```
ftpcompress <name string>, <output name variable>;
```

Some examples are

Exhibit 4.12. table title

```
ftpcompress "www_folder", ~zip_file_name; #compress a folder and all its  
contents
```

```
ftpcompress "index.txt"; #compress a file
```

A compressed file can be uncompressed using the **ftpuncompress** command. The name string specifies the file to be uncompressed. The *zip*, *gz*, *tar.gz*, and *tgz* compressed file formats are supported. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.13. table title

```
ftpuncompress <name string>;
```

Some examples are

Exhibit 4.14. table title

```
ftpuncompress "in22.tar.gz"; #uncompress a tar.gz file
```

```
ftpuncompress "dw441.zip"; #uncompress a zip file
```

Copying and moving local files and folders

Files and folders can be copied using the **ftpcopylocal** command. The item identified by the source name is copied to the specified destination path. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.15. table title

```
ftpcopylocal <source name>, <destination path>;
```

Some examples are

Exhibit 4.16. table title

```
ftpcopylocal "filelist.txt", "C:\\\\backup"; #copy a file  
ftpcopylocal "dw_folder", "C:\\\\backup"; #copy a folder
```

Files and folders can be moved using the **ftpmovelocal** command. The item identified by the source name is moved to the specified destination path. The **ftpresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 4.17. table title

```
ftpmovelocal <source name>, <destination path>;
```

Some examples are

Exhibit 4.18. table title

```
ftpmovelocal "dirlist.txt", "C:\\\\backup"; #move a file  
ftpmovelocal "ac_folder", "C:\\\\backup"; #move a folder
```

5

Reading and writing local files

Opening and closing local files	35
Reading lines	36
Writing lines	36

Opening and closing local files

A file can be opened for reading or writing using the **fileopen** command. The predefined keywords **read**, **write**, and **append** are used to open the file for reading, writing, or appending. If the file is opened for writing, any existing file with the same name is deleted and a new file will be created. The name string specifies the name of the file to open. The **fileresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 5.1. table title

```
fileopen <keywords: read, write, append>, <name string>;
```

Some examples are

Exhibit 5.2. table title

```
fileopen write, "error.txt"; #create or open a new file for writing  
fileopen append, "output.txt"; #open a file to append data to the end of  
the file
```

An opened file can be closed with the **fileclose** command. Since only one file can be opened at a time, no arguments are required.

The syntax is

Exhibit 5.3. table title

```
fileclose;
```

Some examples are

Exhibit 5.4. table title

```
fclose; #close the file
```

Reading lines

The **filereadline** command can be used to read lines from a file that has been opened with the **fileopen** command in read mode. A line of text is read from the file and placed in the user specified variable. The **fileresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 5.5. table title

```
filereadline <user specified variable>;
```

Some examples are

Exhibit 5.6. table title

```
fileopen read, "eg_12.txt";  
filereadline ~one_line; #read a line from the file  
  
filereadline ~one_line; #read the next line in the file
```

Writing lines

The **filewriteline** command can be used to write lines from a file that has been opened with the **fileopen** command in write or append mode. The line specified in the text line string is written to the file. The **fileresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 5.7. table title

```
filewriteline <text line string>;
```

Some examples are

Exhibit 5.8. table title

```
fileopen append, "out_log.txt";  
filewriteline "File 1 was uploaded"; #write a line to the file  
  
filewriteline "File 2 was uploaded"; #write the next line to the file
```

6

Email notification

Creating email messages	39
Multi line messages and attachments	39
Sending email	40

Creating email messages

The **mailcreate** command is used to create an email message that can be sent to recipients using the **mailsend** command. The sender's name and email address should be specified in addition to the email's subject field and the email message that needs to be sent. The **mailresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 6.1. table title

```
mailcreate <sender name>, <sender email address>, <email subject>, <email message>;
```

Some examples are

Exhibit 6.2. table title

```
mailcreate "John Doe", "john@jdinc.com", "Script status", "Upload successful!"; #create email message
```

Multi line messages and attachments

The **mailaddline** command is used to add an additional line to the message body that was created using the **mailcreate** command. The **mailresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 6.3. table title

```
mailaddline <additonal message line>;
```

Some examples are

Exhibit 6.4. table title

```
mailcreate "John Doe", "john@jdinc.com", "Script status", "Upload
  successful!"; #create email message

mailaddline "additional message line"; #add an additional line to the
  email message
```

The **mailaddfile** command is used to attach a file to the message body that was created using the **mailcreate** command. The **mailresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

The syntax is

Exhibit 6.5. table title

```
mailaddfile <path to file to be attached>;
```

Some examples are

Exhibit 6.6. table title

```
mailcreate "John Doe", "john@jdinc.com", "Script status", "Upload
  successful!"; #create email message

mailaddfile "c:\\temp\\status.txt"; #attach a file to the email message
```

Sending email

The **mailsend** command is used to send an email message that has been created using the **mailcreate** command. The same email message can be sent to multiple recipients by calling this command multiple times with the email address of each recipient. The internet address of the mail server and the corresponding port number must be specified in addition to the recipient's name and email address. In most cases, these values can be obtained from your default email program. The **mailresult** predefined status flag is set to the predefined constant **success** if the command completed successfully.

An optional username and password may also be supplied if the mail server requires it. Many mail servers now implicitly require users to authenticate before sending mail. In these cases, the username and password used to read email from this server should be supplied.

The syntax is

Exhibit 6.7. table title

```
mailsend <mail server address>, <server port>, <recipient name>,  
<recipient email> [, optional: <username>, <password>];
```

Some examples are

Exhibit 6.8. table title

```
mailsend "mail.jdinc.com", 25, "Jim Jones", "jj@jj.com", "username",  
"pass"; #send email message  
  
mailsend "mail.jdinc.com", 25, "Joe Smith", "js@js.com", "username",  
"pass"; #send same message to another recipient
```

7

Error reporting

Setting program exit codes	43
Writing out error files	43
Email notifications	43

Setting program exit codes

The **setexitcode** command sets an exit code for the program that is passed back to the command shell on program termination. This is used to report error codes when running ftpshell scripts from the command line.

The syntax is

Exhibit 7.1. table title

```
setexitcode <integer number>;
```

Some examples are

Exhibit 7.2. table title

```
setexitcode 5;
```

Writing out error files

The section on reading and writing local files contains detailed information on opening and writing to a file.

An example for writing out an error file is

Exhibit 7.3. table title

```
fileopen write, "c:\\errorlog.txt";  
filewriteline "An error has occurred";  
fileclose;
```

Email notifications

The section on email notification contains detailed information on creating and sending email.

An example for sending out an email is

Exhibit 7.4. table title

```
mailcreate "John Doe", "support@jdoe.com", "Script status regd.", "Failed  
to upload file.";  
mailsend "mail.jdoe.com", 25, "Joe Smith", "joe@js.com", "username",  
"password";
```

8

Working with timestamps

Generating timestamps	46
Using timestamps	46

Generating timestamps

The **gettimestamp** command is used to generate the current timestamp in YYYYMMDDhhmmss format (Y=year, M=month, D=day, h=hour, m=minute, s=second). The generated timestamp is copied to the user variable. The predefined keywords **incr** and **decr** can be optionally used to generate timestamps for previous or future dates. The predefined keywords **second**, **minute**, **hour**, **day**, **week**, **month**, and **year** are used to specify the time interval to adjust, while the interval count specifies the number of such intervals.

The syntax is

Exhibit 8.1. table title

```
gettimestamp <user variable> [, optional: <keywords: incr, decr>,  
<keywords: second, minute, hour, day, week, month, year>, <interval  
count>];
```

Some examples are

Exhibit 8.2. table title

```
gettimestamp ~my_timestamp; #get current timestamp  
  
gettimestamp ~my_timestamp, decr, month, 2; #get a timestamp that is 2  
months old  
  
gettimestamp ~my_timestamp, incr, day, 5; #get a timestamp that is 5 days  
in the future
```

Using timestamps

Timestamps or portions of timestamps are frequently added to file names. The following examples show how time stamps can be appended to files and how file names can be searched for specific timestamps.

Exhibit 8.3. table title

```
gettimestamp ~my_timestamp; # generate current timestamp

foreach $local_item in @local_list begin
  strprint ~new_name, ~my_timestamp, "_", $item.name; #prepend timestamp
  to file name
end

gettimestamp ~my_timestamp, decr, day, 1; #generate timestamp that is a
day old

foreach $local_item in @local_list begin
  strprint ~search_name, "*", ~my_timestamp, "*"; #generate search string
  if ~search_name eq $local_item.name begin
    #search for files with this timestamp as part of their filename
  end
end
```

9

Accessing system information

Reading environmental variables	49
Getting the local computer name	49

Reading environmental variables

The **getenvvar** command is used to access the values of environmental variables. The user variable receives the value of the specified environmental variable string. Refer to the section on command line script parameters for a method to create and set program specific environmental variables from the command line.

The syntax is

Exhibit 9.1. table title

```
getenvvar <user variable>, <environment variable string>;
```

An example is

Exhibit 9.2. table title

```
getenvvar ~os_variable, "OS"; #get the value of the environmental variable  
OS
```

Getting the local computer name

The **getcompname** command gets the name of the computer that the program is running on. The computer name is copied to the specified user variable.

The syntax is

Exhibit 9.3. table title

```
getcompname <user variable>;
```

An example is

Exhibit 9.4. table title

```
getcompname ~comp_name; #get the name of this computer
```

10

Command line script parameters

Running a script from the command line	52
Specifying log files	52
Passing Environmental variables	52

Running a script from the command line

An ftpshell script can be run from the command line using the `-script` switch.

Exhibit 10.1. table title

```
getcompname <user variable>;
```

Specifying log files

The following switches are related to log file generation.

<code>-logfile</code>	generate an output log file
<code>-tslogfile</code>	generate a time stamped output log file
<code>-append</code> (default is overwrite)	append to log file generated during previous run
<code>-append <max file size></code>	roll over the log file after the maximum file size specified in bytes

Some examples are

Exhibit 10.2. table title

```
ftpshell -script "test.fscr" -tslogfile "test.log" #generate a time
stamped log file

ftpshell -script "test.fscr" -logfile "test.log" -append 200000 #rollover
the appended log file every 200000 bytes
```

Passing Environmental variables

The following switch enables environmental variables to be created and passed to the program.

`-set <environment variable>=<value>` create and set an environmental variable

Some examples are

Exhibit 10.3. table title

```
ftpshell -script "run.fscr" -set PASSWORD="mypass" #create env variable  
PASSWORD and set it to mypass
```

```
ftpshell -script "run.fscr" -set SERV_IP="121.122.12.1" #pass env  
variable. Value can be extracted from the script using getenvvar
```

Index

- A**
append, 35
ascii, 26
auto, 26
- B**
begin, 5
binary, 26
bydate, 24
bysize, 24
- C**
certload, 19
- D**
day, 46
decr, 46
decrement, 12
disablepasv, 25
- E**
else, 4
enablepasv, 25
end, 5
endscript, 9
exitloop, 8
- F**
file, 23, 23, 29
fileclose, 35
fileopen, 35, 36, 36
filereadline, 36
fileresult, 7
filewriteline, 36
folder, 23, 23, 29
foreach, 3, 6
ftpcompress, 31
ftpconnect, 19
ftpconnectssh, 20, 20
ftpconnectssl, 19, 19
ftpconnectsslc, 19
ftpcopylocal, 33
ftpcustomcmd, 30
ftpdelete, 29
ftpdconnect, 20
ftpdownload, 23
ftpgetlist, 3, 6, 22
ftpmakefolder, 30
ftpregexmatch, 16
ftpmovevlocal, 33
ftprename, 29
ftpresult, 3, 7, 16, 16, 19, 21, 22, 23, 23, 29, 29, 30, 30, 31, 31, 32, 33, 33, 35, 36, 36
ftprunprogram, 31
ftpsetpath, 21, 23, 23
ftpuncompress, 32
ftpupload, 23
ftpwildcardmatch, 16
- G**
getcompname, 49
getenvvar, 49
gettimestamp, 46
- H**
hour, 46
- I**
if, 4
in, 6
incr, 46
increment, 12
- L**
left, 13
local, 21, 22, 29, 29, 30
loop, 5
- M**
mailaddfile, 40
mailaddline, 39
mailcreate, 39, 39, 40, 40
mailresult, 3, 7, 39, 39, 40, 40
mailsend, 39, 40

minute, 46
month, 46

O

overwrite, 24

P

pkeyload, 20
print, 11
proxyhttp, 20

R

read, 35
regexreplace, 14
remote, 21, 22, 29, 29, 30
rename, 24
resume, 24
right, 13

S

second, 46
setduperules, 24
setexitcode, 43
setnlst, 26
settransfertype, 26
setvar, 11
skip, 24
strlower, 12
strprint, 11
strslice, 13
strupper, 12

U

unsetnlst, 26

W

waitsecs, 8
week, 46
write, 35

Y

year, 46