FOR

Windows NT 4/2000/XP Syntax

Note: The parts of this text that are displayed in magenta are valid for Windows 2000 and XP only

Runs a specified command for each file in a set of files.

```
FOR  %variable IN (set) DO command [command-parameters]
```

`%variable`	Specifies a replaceable parameter.
`(set)`	Specifies a set of one or more files. Wildcards may be used.
`command`	Specifies the command to carry out for each file.
`command-parameters`	Specifies parameters or switches for the specified command.

To use the FOR command in a batch program, specify `%%variable` instead of `%variable`
Variable names are case sensitive, so %i is different from %I.

If Command Extensions are enabled, the following additional forms of the FOR command are supported:

```
FOR  /D %variable IN (set) DO command [command-parameters]
```

If *set* contains wildcards, then specifies to match against directory names instead of file names.

```
FOR  /R [[drive:]path] %variable IN (set) DO command [command-parameters]
```

Walks the directory tree rooted at [*drive:*]*path*, executing the FOR statement in each directory of the tree.
If no directory specification is specified after /R then the current directory is assumed.
If set is just a single period (.) character then it will just enumerate the directory tree.

```
FOR  /L %variable IN (start,step,end) DO command [command-parameters]
```

The *set* is a sequence of numbers from *start* to *end*, by *step* amount.
So (1,1,5) would generate the sequence 1 2 3 4 5 and (5,-1,1) would generate the sequence (5 4 3 2 1)

Note: If *step* equals 0, and *end* is greater than or equal to *start*, then the loop will continue forever:

```
FOR /L %A IN (0,0,0) DO command [command-parameters]
```

This is the batch equivalent of `Do Forever` (continuous) loops in "real" scripting languages.

```
FOR  /F ["options"] %variable IN (filenameset) DO command [command-parameters]

FOR  /F ["options"] %variable IN ("string") DO command [command-parameters]

FOR  /F ["options"] %variable IN ('command') DO command [command-parameters]
```

or, if usebackq option present:

```
FOR  /F ["options"] %variable IN (filenameset) DO command [command-parameters]

FOR  /F ["options"] %variable IN ('string') DO command [command-parameters]

FOR  /F ["options"] %variable IN (`command`) DO command [command-parameters]
```

filenameset is one or more file names.
Each file is opened, read and processed before going on to the next file in *filenameset.*
Processing consists of reading in the file, breaking it up into individual lines of text and then parsing each line into zero or more tokens. The body of the for loop is then called with the variable value(s) set to the found token string(s).
By default, /F passes the first blank separated token from each line of each file.
Blank lines are skipped.
You can override the default parsing behavior by specifying the optional "options" parameter. This is a quoted string which contains one or more keywords to specify different parsing parameters.
The keywords are:

`eol=`*c*	-	specifies an end of line comment character (just one).
		Note: The default `eol` character is the semicolon (`;`). That is why `FOR /F` loops skip lines starting with semicolons *unless* a different `eol` character is specified (try `"eol="`).
`skip=`*n*	-	specifies the number of lines to skip at the beginning of the file.
`delims=`*xxx*	-	specifies a delimeter set. This replaces the default delimiter set of space and tab.
`tokens=`*x,y,m-n*	-	specifies which tokens from each line are to be passed to the for body for each iteration. This will cause additional variable names to be allocated. The *m-n* form is a range, specifying the mth through the nth tokens. If the last character in the tokens= string is an asterisk, then an additional variable is allocated and receives the remaining text on the line after the last token parsed.
`usebackq`	-	specifies that the new semantics are in force, where a back quoted string is executed as a command and a single quoted string is a literal string command and allows the use of double quotes to quote file names in *filenameset.*

Some examples might help:

```
FOR  /F "eol=; tokens=2,3* delims=, " %i in (myfile.txt) do @echo %i %j %k
```

would parse each line in myfile.txt, ignoring lines that begin with a semicolon, passing the 2nd and 3rd token from each line to the for body, with tokens delimited by commas and/or spaces.
Notice the for body statements reference %i to get the 2nd token, %j to get the 3rd token, and %k to get all remaining tokens after the 3rd.
For file names that contain spaces, you need to quote the filenames with double quotes.
In order to use double quotes in this manner, you also need to use the usebackq option, otherwise the double quotes will be interpreted as defining a literal string to parse.

%i is explicitly declared in the for statement and the %j and %k are implicitly declared via the tokens= option.
You can specify up to 26 tokens via the tokens= line, provided it does not cause an attempt to declare a variable higher than the letter 'z'.
Remember, FOR variable names are global, and you can't have more than 26 (NT 4) or 52 (Windows 2000/XP) total active at any one time.

You can also use the FOR /F parsing logic on an immediate string, by making the filenameset between the parenthesis a quoted string. It will be treated as a single line of input from a file and parsed.

Finally, you can use the FOR /F command to parse the output of a command. You do this by making the filenameset between the parenthesis a single quoted (NT 4) or back quoted (Windows 2000/XP) string. It will be treated as a command line, which is passed to a child CMD.EXE and the output is captured into memory and parsed as if it was a file.
So the following examples:

```
FOR  /F "delims==" %i IN ('set') DO @echo %i

FOR  /F "usebackq delims==" %i IN (`set`) DO @echo %i
```

would enumerate the environment variable names in the current environment.

In addition, substitution of FOR variable references has been enhanced.
You can now use the following optional syntax:

`%~i`	-	expands %i removing any surrounding quotes (")
`%~fi`	-	expands %i to a fully qualified path name
`%~di`	-	expands %i to a drive letter only
`%~pi`	-	expands %i to a path only
`%~ni`	-	expands %i to a file name only
`%~xi`	-	expands %i to a file extension only
`%~si`	-	expanded path contains short names only
`%~ai`	-	expands %i to file attributes of file
`%~ti`	-	expands %i to date/time of file
`%~zi`	-	expands %i to size of file
`%~$PATH:i`	-	searches the directories listed in the PATH environment variable and expands %i to the fully qualified name of the first one found. If the environment variable name is not defined or the file is not found by the search, then this modifier expands to the empty string

The modifiers can be combined to get compound results:

`%~dpi`	-	expands %i to a drive letter and path only
`%~nxi`	-	expands %i to a file name and extension only
`%~fsi`	-	expands %i to a full path name with short names only
`%~dp$PATH:i`	-	searches the directories listed in the PATH environment variable for %i and expands to the drive letter and path of the first one found.
`%~ftzai`	-	expands %i to a DIR like output line

In the above examples %i and PATH can be replaced by other valid values.
Just be careful to pick your FOR variable letters to not conflict with any of the format specifier letters if you plan on using the enhanced substitution logic.
The %~ syntax is terminated by a valid FOR variable name.
Picking upper case variable names like %I makes it more readable and avoids confusion with the modifiers, which are not case sensitive.