Table of Contents

Using 4NT

Using the Command Line
File Selection
Directory Navigation
Other Features

Commands

Commands by Category
Commands by Name

Aliases and Batch Files

Aliases
Batch Files

The Environment

Environment Variables and Functions
Internal Variables
Variable Functions

Configuring 4NT

4NT.INI

Setup and Troubleshooting

Starting 4NT
What's New?
The 4NT Help System
Error Messages
Troubleshooting, Service, and Support

Reference Information

File Systems and File Name Conventions
Miscellaneous Reference Information
ASCII and Key Codes
Glossary
Copyright and Version

COPYRIGHT and VERSION

4NT

Version 3.00 Help System

Text by Hardin Brothers, Tom Rawson, and Rex Conn

[07/97 - 3.00B]

The Command Line

4NT displays a **[c:\]** prompt when it is waiting for you to enter a command. (The actual text depends on the current drive and directory as well as your PROMPT settings.) This is called the command line and the prompt is asking you to enter a command, an alias or batch file name, or the instructions necessary to begin an application program.

This section explains the features that will help you while you are typing in commands, and how keystrokes are interpreted when you enter them at the command line. The keystrokes discussed here are the ones normally used by 4NT. If you prefer using different keystrokes to perform these functions, you can assign new ones with key mapping directives in the *.INI* file.

The command line features documented in this section are:

Command-Line Editing

Command History and Recall

Command History Window

Filename Completion

Automatic Directory Changes

Directory History Window

Multiple Commands

Expanding and Disabling Aliases

Command-Line Length Limits

Additional command-line features are documented under File Selection and under Directory Navigation.

Command-Line Editing

The command line works like a single-line word processor, allowing you to edit any part of the command at any time before you press **Enter** to execute it, or **Esc** to erase it. The command line you enter can be up to 1023 characters long.

You can use the following editing keys when you are typing a command (the words **Ctrl** and **Shift** mean to press the Ctrl or Shift key together with the other key named):

Cursor Movement Keys:

←	Move the cursor left one character.
→	Move the cursor right one character.
Ctrl ←	Move the cursor left one word.
Ctrl →	Move the cursor right one word.
Home	Move the cursor to the beginning of the line.
End	Move the cursor to the end of the line.

Insert and Delete:

Ins	Toggle between insert and overstrike mode.
Del	Delete the character at the cursor.
Bksp	Delete the character to the left of the cursor.
Ctrl-L	Delete the word or partial word to the left of the cursor.
Ctrl-R or **Ctrl-Bksp**	Delete the word or partial word to the right of the cursor.
Ctrl-Home	Delete from the beginning of the line to the cursor.
Ctrl-End	Delete from the cursor to the end of the line.
Esc	Delete the entire line.

Execution:

Ctrl-C or **Ctrl-Break**	Cancel the command line.
Enter	Execute the command line.

Most of the command-line editing capabilities are also available when a 4NT command prompts you for a line of input. For example, you can use the command-line editing keys when DESCRIBE prompts for a file description, when INPUT prompts for input from an alias or batch file, or when LIST prompts you for a search string.

If you want your input at the command line to be in a different color from 4NT's prompts or output, you can use the Display page of the OPTION dialogs, or the InputColors directive in your *.INI* file.

4NT will prompt for additional command-line text when you include the escape character as the very last character of a typed command line. The default escape character is the caret [**^**]. For example:

```
[c:\] echo The quick brown fox jumped over the lazy ^
More? sleeping dog. > alphabet
```

Sometimes you may want to enter one of the command line editing keystrokes on the command line, instead of performing the key's usual action. For example, suppose you have a program that requires a Ctrl-R character on its command line. Normally you couldn't type this keystroke at the prompt, because it would be interpreted as a "Delete word right" command.

To get around this problem, use the special keystroke **Alt-255**. You enter Alt-255 by holding down the **Alt** key while you type **255** on the numeric keypad, then releasing the **Alt** key (you must use the number keys on the numeric pad; the row of keys at the top of your keyboard won't work). This forces 4NT to interpret the next keystroke literally and places it on the command line, ignoring any special meaning it would normally have as a command- line editing or history keystroke. You can use Alt-255 to suppress the normal meaning of command-line editing keystrokes even if they have been reassigned with key mapping directives in the *.INI* file, and Alt-255 itself can be reassigned with the CommandEscape directive.

Command History and Recall

Command History Keys:

	Recall the previous (or most recent) command, or the most recent command that matches a partial command line.
↓	Recall the next (or oldest) command, or the oldest command that matches a partial command line.
F3	Fill in the rest of the command line from the previous command, beginning at the current cursor position.
Ctrl-D	Delete the currently displayed history list entry, erase the command line, and display the previous (matching) history list entry.
Ctrl-E	Display the last entry in the history list.
Ctrl-K	Save the current command line in the history list without executing it, and then clear the command line.
Ctrl-Enter	Copy the current command line to the end of the history list even it has not been altered, then execute it.
@	As the first character in a line: Do not store the current line in the CMDLINE environment variable.

Use the key repeatedly to scan back through the history list. When the desired command appears, press **Enter** to execute it again. After you have found a command, you can edit it before pressing **Enter**.

The history list is normally "circular". If you move to the last command in the list and then press the down arrow one more time, you'll see the first command in the list. Similarly, if you move to the first command in the list and then press the up arrow one more time, you'll see the last command in the list. You can disable this feature and make command history recall stop at the beginning or end of the list by turning off the History Wrap selection on the Command Line 1 page of the OPTION dialogs, or setting HistWrap to No in the *.INI* file.

You can search the command history list to find a previous command quickly using **command completion**.

Just enter the first few characters of the command you want to find and press . You only need to enter enough characters to identify the command that you want to find. If you press the key a second time, you will see the previous command that matches. The system will beep if there are no matching commands. The search process stops as soon as you type one of the editing keys, whether or not the line is changed. At that point, the line you're viewing becomes the new line to match if you press again.

You can specify the size of the command history list on the Command Line 1 page of the OPTION dialogs, or with the History directive in the *.INI* file. When the list is full, the oldest commands are discarded to make room for new ones. You can also use the HistMin directive in the *.INI* file to enable or disable history saves and to specify the shortest command line that will be saved.

You can prevent any command line from being saved in the history by beginning it with an at sign [**@**].

When you execute a command from the history, that command remains in the history list in its original position. The command is not copied to the end of the list (unless you modify it). If you want each command to be copied or moved to the end of the list when it is re-executed, set HistCopy or HistMove to Yes in your *.INI* file. If you select either of these options, the list entry identified as "current" (the entry from which commands are retrieved when you press) is also adjusted to refer to the end of the history list after each recalled command is executed.

Local and Global Command History

The command history can be stored in either a "local" or "global" list.

With a local command history list, any changes made to the history will only affect the current copy of 4NT.   They will not be visible in other shells, or other sessions.

With a global command history list, all copies of 4NT will share the same command history, and any changes made to the history in one copy will affect all other copies.   Global lists are the default for 4NT.

You can control the type of command history list on the Startup page of the OPTION command, with the LocalHistory directive in the *.INI* file, and the **/L** and **/LH** options of the START command.

If you select a global command history list for 4NT you can share the history among all copies of 4NT running in any session.   When you close all 4NT sessions, the memory for the global history list is released, and a new, empty history list is created the next time you start 4NT.

If you want the command history list to be retained in memory even when no command processor session is running, execute the SHRALIAS command, which loads a program to perform this service for global command history, directory history, and alias lists.

SHRALIAS retains the alias list in memory, but cannot preserve it when Windows NT itself is shut down. To save your aliases when restarting NT, you must store them in a file and reload them after the system restarts.   For details on how to do so, see the HISTORY command.

Whenever you start a secondary shell which uses a local history list, it inherits a copy of the command history from the previous shell.   However, any changes to the history made in the secondary shell will affect only that shell.   If you want changes made in a secondary shell to affect the previous shell, use a global history list in both shells.

Command History Window

Command History Window Keys:

PgUp or **PgDn**	(from the command line) Open the command history window.
	Scroll the display up one line.
↓	Scroll the display down one line.
←	Scroll the display left 4 columns.
→	Scroll the display right 4 columns.
PgUp	(inside the window) Scroll the display up one page.
PgDn	(inside the window) Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the history list.
Ctrl-PgDn or **End**	Go to the end of the history list.
Ctrl-D	Delete the selected line from the history list.
Enter	Execute the selected line.
Ctrl-Enter	Move the selected line to the command line for editing.

You can view the command history in a scrollable **command history window**, and select the command to modify or re-execute from those displayed in the window. To activate the command history window press **PgUp** or **PgDn** at the command line. A window will appear in the upper right corner of the screen, with the command you most recently executed marked with a highlight. (If you just finished re-executing a command from the history, then the next command in sequence will be highlighted.)

Once you have selected a command in the history window, press **Enter** to execute it immediately, or **Ctrl-Enter** to move the line to the prompt for editing (you cannot edit the line directly in the history window).

You can bring up a "filtered" history window by typing some characters on the command line, then pressing PgUp or PgDn. Only those commands matching the typed characters will be displayed in the window.

See Popup Windows for information on customizing window position, size, and color.

Filename Completion

Filename Completion Keys:

F8 or **Shift-Tab**	Get the previous matching filename.
F9 or **Tab**	Get the next matching filename.
F10	Keep the current matching filename and display the next matching name immediately after the current one.
Ctrl-A	On LFN drives, toggle between long filename and short filename format.

Filename completion can help you by filling in a complete file name on the command line when you only remember or want to type part of the name. For example, if you know the name of a file begins *AU* but you can't remember the rest of the name, type:

```
[c:\] copy au
```

and then press the **Tab** key or **F9** key. 4NT will search the current directory for filenames that begin *AU* and insert the first one onto the command line in place of the *AU* that you typed.

If this is the file that you want, simply complete the command. If 4NT didn't find the file that you were looking for, press **Tab** or **F9** again to substitute the next filename that begins with *AU*. When there are no more filenames that match your pattern, the system will beep each time you press **Tab** or **F9**.

If you go past the filename that you want, press **Shift-Tab** or **F8** to back up and return to the previous matching filename. After you back up to the first filename, the system will beep each time you press **Shift-Tab** or **F8**.

If you want to enter more than one matching filename on the same command line, press **F10** when each desired name appears. This will keep that name and place the next matching filename after it on the command line. You can then use **Tab** (or **F9**), **Shift-Tab** (or **F8**), and **F10** to move through the remaining matching files.

Typing **Ctrl-A** on the command line during filename expansion on a FAT drive will toggle the returned filename between long filename (LFN) and the traditional short name (SFN) formats. The default is LFN format; if you switch to SFN format, the change will only remain in effect for the current filename expansion. Any new expansion sequence later on the command line will start in LFN format and can be toggled to SFN format with another **Ctrl-A**.

The pattern you use for matching may contain any valid filename characters, as well as wildcard characters and extended wildcards. For example, you can copy the first matching *.TXT* file by typing

```
[c:\] copy *.txt
```

and then pressing **Tab**.

If you don't specify part of a filename before pressing **Tab**, the command processor will match all files. For example, if you enter the above command as "COPY ", without the "*.TXT", and then press **Tab**, the first filename in the current directory is displayed. Each time you press **Tab** or **F9** after that, another name from the current directory is displayed, until all filenames have been displayed.

If you type a filename without an extension, 4NT will add **.** to the name (*** on LFN, HPFS, and NTFS drives). It will also place a "*" after a partial extension. If you are typing a group of file names in an include list, the part of the include list at the cursor will be used as the pattern to match.

When filename completion is used at the start of the command line, it will only match directories, executable files, and files with executable extensions, since these are the only file names that it makes sense to use at the start of a command. If a directory is found, a "\" will be appended to it to enable an automatic directory change.

Converting Between Long and Short Filenames

On LFN drives, 4NT will search for and display long filenames during filename completion. If you want to search for traditional 8.3 short filenames, press **Ctrl-A** before you start using filename completion. This allows you to use filename completion on LFN drives with applications that do not support long filenames.

You can press **Ctrl-A** at any time prior to beginning filename completion. The switch to the short filename format remains in effect for the remainder of the current command line. When the command processor begins a new command line it will return to long filename format unless you press **Ctrl-A** again.

You can also press **Ctrl-A** just after a filename is displayed, and the name will be converted to short filename format**.** However, this feature only affects the most recently entered file or directory name (the part between the cursor and the last backslash [\] on the command line), and any subsequent entries. It will not automatically convert all the parts of a previously entered path.

Ctrl-A "toggles" the filename completion mode, so you can switch back and forth between long and short filename displays by pressing **Ctrl-A** each time you want to change modes.

Several topics are related to filename completion:

Appending Backslashes to Directory Names

Customizing Filename Completion

Filename Completion Window

Appending Backslashes to Directory Names

If you set the AppendToDir *.INI* directive, or the "Add \ ..." option on Command Line 1 page of the OPTION dialogs, 4NT will add a trailing backslash [\] to all directory names.   This feature can be especially handy if you use filename completion to specify files that are not in the current directory -- a succession of **Tab** (or **F9**) and **F10** keystrokes can build a complete path to the file you want to work with.

The following example shows the use of this technique to edit the file *C:\DATA\FINANCE\MAPS.DAT*. The lines which include *<F9>* show where **F9** (or **Tab**) is pressed; the other lines show how the command line appears after the previous **F9** or **Tab** (the example is displayed on several lines here, but all appears at a single command prompt when you actually perform the steps):

```
1      [c:\] edit \da <F9>
2      [c:\] edit \data\
3      [c:\] edit \data\f <F9>
4      [c:\] edit \data\frank.doc <F9>
5      [c:\] edit \data\finance\
6      [c:\] edit \data\finance\map <F9>
7      [c:\] edit \data\finance\maps.dat
```

Note that **F9** was pressed twice in succession on lines 3 and 4, because the file name displayed on line 3 was not what was needed — we were looking for the FINANCE directory, which came up the second time **F9** was pressed.   In this example, filename completion saves about half the keystrokes that would be required to type the name in full.   If you are using long file or directory names, the savings can be much greater.

Customizing Filename Completion

You can customize filename completion for any internal or external command or alias.   This allows the command processor to display filenames intelligently based on the command you are entering.   For example, you might want to see only *.TXT* files when you use filename completion in the EDIT command.

To customize filename completion you can use the Command Line 1page of the OPTION dialogs, or set the FileCompletion directive manually in your *.INI* file.   You can also use the FILECOMPLETION environment variable.   If you use both, the environment variable will override the settings in your *.INI* file. You may find it useful to use the environment variable for experimenting, then create permanent settings with the OPTION command or the FileCompletion directive.

The format for both the environment variable and the *.INI* file is:

```
cmd1:ext1 ext2 ...; cmd2: ...
```

where "cmd" is a command name and "ext" is a file extension (which may include wildcards) or one of the following file types:

```
DIRS        Directories
RDONLYRead-only files
HIDDENHidden files
SYSTEMSystem files
ARCHIVE     Files modified since the last backup
```

The command name is the internal command, alias command, or executable file name (without a path). For example, to have file completion return only directories for the CD command and only *.C* and *.ASM* files for B (the Boxer editor), you would use this setting for filename completion in the OPTION dialogs:

```
FileCompletion=cd:dirs; b:c asm
```

To set the same values using the environment variable, you would use this line:

```
[c:\] set filecompletion=cd:dirs; b:c asm
```

With this setting in effect, if you type "CD " and then pressed **Tab**, the command processor will return only directories, not files.   If you type "B " and press **Tab**, you will see only names of *.C* and *.ASM* files.

4NT does **not** check your command line for aliases before matching the commands for customized file completion.   Instead, they ignore any path or file extension information in the first word of the command, and then search the FILECOMPLETION environment variable and the FileCompletion *.INI* directive to find a match that will limit the files selected for filename completion.

Filename Completion Window

You can also view filenames in a **filename completion window** and select the file you want to work with. To activate the window, press **F7** or **Ctrl-Tab** at the command line. You will see a window in the upper-right corner of the screen, with a sorted list of files that match any partial filename you have entered on the command line. If you haven't yet entered a file name, the window will contain the name of all files in the current directory. You can search for a name by typing the first few characters. (**Ctrl-Tab** will work only if your keyboard and keyboard driver support it. If it does not work on your system, use **F7** instead.)

Filename Completion Window Keys:

F7 or **Ctrl-Tab**	(from the command line) Open the filename completion window.
	Scroll the display up one line.
↓	Scroll the display down one line.
←	Scroll the display left 4 columns.
→	Scroll the display right 4 columns.
PgUp	Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the filename list.
Ctrl-PgDn or **End**	Go to the end of the filename list.
Enter	Insert the selected filename into the command line.

See Popup Windows for information on customizing window position, size, and color.

Automatic Directory Changes

The automatic directory change feature lets you change directories quickly from the command prompt, without entering an explicit CD or CDD command. To do so, simply type the name of the directory you want to change to at the prompt, with a backslash [\] at the end. For example:

```
[c:\] 4NT\
[c:\4NT]
```

This can make directory changes very simple when it is combined with Extended Directory Searches or CDPATH. If you have enabled either of those features, the command processor will use them in searching for any directory you change to with an automatic directory change.

For example, suppose Extended Directory Searches are enabled, and the directory WIN exists on drive E:. You can change to this directory with a single word on the command line:

```
[c:\4NT] win\
[e:\WIN]
```

(Depending on the way Extended Directory Searches are configured, and the number of subdirectories on your disk whose names contain the string *WIN*, when you execute such a command you may see an immediate change as shown above, or a popup window which contains a list of subdirectories named *WIN* to choose from.)

The text before the backslash can include a drive letter, a full path, a partial path, or a UNC name. Commands like "....\" can be used to move up the directory tree quickly (see Extended Parent Directory Names). Automatic directory changes save the current directory, so it can be recalled with a "CDD -" or "CD -" command. For example, any of the following are valid automatic directory change entries:

```
[c:\] d:\data\finance\
[c:\] archives\
[c:\] ...\util\windows\
[c:\] \\server\vol1\george\
```

The first and last examples change to the named directory. The second changes to the *ARCHIVES* subdirectory of the current directory, and the third changes to the *UTIL\WINDOWS* subdirectory of the directory which is two levels "up" from the current directory in the tree.

Directory History Window

Directory History Window Keys:

Ctrl-PgUp or **Ctrl-PgDn**	(from the command line) Open the directory history window.
	Scroll the display up one line.
↓	Scroll the display down one line.
←	Scroll the display left 4 columns.
→	Scroll the display right 4 columns.
PgUp	Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the directory list.
Ctrl-PgDn or **End**	Go to the end of the directory list.
Ctrl-D	Delete the selected line from the directory list.
Enter	Change to the selected drive and directory.
Ctrl-Enter	Move the selected line to the command line for editing.

The current directory is recorded automatically in the **directory history list** just before each change to a new directory or drive.

You can view the directory history from the **directory history window** and change to any drive and directory on the list. To activate the directory history window, press **Ctrl-PgUp** or **Ctrl-PgDn** at the command line.   You can then select a new directory with the **Enter** key.

If the directory history list becomes full, old entries are deleted to make room for new ones.   You can set the size of the list with the DirHistory directive in the *.INI* file.   In order to conserve space, each directory name is recorded just once in the directory history, even if you move into and out of that directory several times.   The directory history can be stored in either a "local" or "global" list.

When you switch directories the original directory is saved in the directory history list, regardless of whether you change directories at the command line, from within a batch file, or from within an alias. However, directory changes made by external directory navigation utilities or other external programs are not recorded by 4NT.

See Popup Windows for information on customizing window position, size, and color.

Local and Global Directory History

The directory history can be stored in either a "local" or "global" list.

With a local directory history list, any changes made to the list will only affect the current copy of the command processor.   They will not be visible in other shells, or other sessions.

With a global list, all copies of the command processor will share the same directory history, and any changes made to the list in one copy will affect all other copies.   Global lists are the default for 4NT.

You can control the type of directory history list on the Startup page of the OPTION dialogs, with the LocalDirHistory directive in the *.INI* file, with the **/L** and **/LD** startup options, and with the **/L** and **/LD** options of the START command

There is no fixed rule for deciding whether to use a local or global directory history list. Depending on your work style, you may find it most convenient to use one type, or a mixture of types in different sessions or shells. We recommend that you start with the default setting (global), then modify it if you find a situation where the default is not convenient.

If you select a global directory list, you can share the list among all copies of 4NT running in any session. When you close all 4NT sessions, the memory for the global directory history list is released, and a new, empty list is created the next time you start 4NT.

If you want the list to be retained in memory even when no command processor session is running, execute the SHRALIAS command, which loads a program to perform this service for the global command history, directory history, and alias lists. SHRALIAS retains the directory history list in memory, but cannot preserve it when Windows NT itself is shut down. 4NT always starts with an empty directory history after the system is restarted.

Whenever you start a secondary shell which uses a local directory history list, it inherits a copy of the directory history from the previous shell. However, any changes to the list made in the secondary shell will affect only that shell. If you want changes made in a secondary shell to affect the previous shell, use a global directory history list in both shells.

Multiple Commands

You can type several commands on the same command line, separated by an ampersand [**&**]. For example, if you know you want to copy all of your *.TXT* files to drive A: and then run CHKDSK to be sure that drive A's file structure is in good shape, you could enter the following command:

```
      [c:\] copy *.txt a: & chkdsk a:
```

You may put as many commands on the command line as you wish, as long as the total length of the command line does not exceed 1023 characters.

You can use multiple commands in alias definitions and batch files as well as from the command line.

If you don't like using the default command separator, you can pick another character using the SETDOS /C command or the CommandSep directive in the *.INI* file. If you plan to share aliases or batch files between 4NT and 4DOS, 4OS2, or Take Command, see Special Character Compatibility for details about choosing compatible command separators for two or more products.

Expanding and Disabling Aliases

A few command line options are specifically related to aliases, and are documented briefly here for completeness.

You can expand an alias on the command line and view or edit the results by pressing **Ctrl-F** before the command is executed.   Doing so is especially useful when you are developing and debugging a complex alias or if you want to make sure that an alias that you may have forgotten won't change the intent of your command.

At times, you may want to temporarily disable an alias that you have defined. To do so, precede the command with an asterisk [*****].   For example, if you have an alias for DIR which changes the display format, you can use the following command to bypass the alias and display the directory in the standard format:

```
[c:\] *dir
```

Command-Line Length Limits

When you first enter a command at the prompt or in an alias or batch file, it can be up to 1,023 characters long.

As 4NT scans the command line and substitutes the contents of aliases and environment variables for their names, the line usually gets longer. This expanded line is stored in an internal buffer which allows each individual command to grow to 1,023 characters during the expansion process. In addition, if you have multiple commands on a single line, during expansion the entire line can grow to as much as 2,047 characters. If your use of aliases or environment variables causes the command line to exceed either of these limits as it is expanded, you will see a "Command line too long" error and the remainder of the line will not be executed.

File Selection

Most internal commands (like COPY, DIR, etc.) work on a file or a group of files.   Besides typing the exact name of the file you want to work with, you can use several shorthand forms for naming or selecting files and the applications associated with them.

Most of the features explained in this section apply to 4NT commands only, and generally can not be used to pass file names to external programs unless those programs were specifically written to support these features.

The file selection features are:

Extended Parent Directory Names

Wildcards

Date, Time, and Size Ranges

File Exclusion Ranges

Multiple Filenames

Include Lists

LFN File Searches

Executable Extensions

Using Windows File Associations

Using Internet URLs

Waiting for Applications to Finish

Extended Parent Directory Names

4NT allows you to extend the traditional syntax for naming the parent directory, by adding additional [**.**] characters. Each additional [**.**] represents an additional directory level above the current directory. For example, *.\FILE.DAT* refers to a file in the current directory, *..\FILE.DAT* refers to a file one level up (in the parent directory), and *...\FILE.DAT* refers to a file two levels up (in the parent of the parent directory). If you are in the *C:\DATA\FINANCE\JANUARY* directory and want to copy the file *LETTERS.DAT* from the directory *C:\DATA* to drive A:

```
[C:\DATA\FINANCE\JANUARY] copy ...\LETTERS.DAT A:
```

Wildcards

Wildcards let you specify a file or group of files by typing a partial filename. The appropriate directory is scanned to find all of the files that match the partial name you have specified.

Wildcards are usually used to specify which files **should** be processed by a command. If you need to specify which files should **not** be processed see File Exclusion Ranges (for internal commands), or EXCEPT (for external commands).

Most internal commands accept filenames with wildcards anywhere that a full filename can be used. There are two wildcard characters, the asterisk [*****] and the question mark [**?**], plus a special method of specifying a range of permissible characters.

An asterisk [*****] in a filename means "any zero or more characters in this position." For example, this command will display a list of all files in the current directory:

```
[c:\] dir *.*
```

If you want to see all of the files with a *.TXT* extension, you could type this:

```
[c:\] dir *.txt
```

If you know that the file you are looking for has a base name that begins with *ST* and an extension that begins with *.D*, you can find it this way. Filenames such as *STATE.DAT*, *STEVEN.DOC*, and *ST.D* will all be displayed:

```
[c:\] dir st*.d*
```

With 4NT, you can also use the asterisk to match filenames with specific letters somewhere inside the name. The following example will display any file with a *.TXT* extension that has the letters *AM* together anywhere inside its base name. It will, for example, display *AMPLE.TXT*, *STAMP.TXT*, *CLAM.TXT*, and *AM.TXT*:

```
[c:\] dir *am*.txt
```

A question mark [**?**] matches any single filename character. You can put the question mark anywhere in a filename and use as many question marks as you need. The following example will display files with names like *LETTER.DOC* and *LATTER.DAT*, and *LITTER.DU*:

```
[c:\] dir l?tter.d??
```

The use of an asterisk wildcard before other characters, and of the character ranges discussed below, are enhancements to the standard wildcard syntax, and may not work properly with software other than 4DOS, 4OS2, 4NT, and Take Command.

"Extra" question marks in your wildcard specification are ignored if the file name is shorter than the wildcard specification. For example, if you have files called *LETTER.DOC*, *LETTER1.DOC*, and *LETTERA.DOC*, this command will display all three names:

```
[c:\] dir letter?.doc
```

The file *LETTER.DOC* is included in the display because the "extra" question mark at the end of "*LETTER?* " is ignored when matching the shorter name *LETTER*.

In some cases, the question mark wildcard may be too general. You can also specify what characters you want to accept (or exclude) in a particular position in the filename by using square brackets. Inside the brackets, you can put the individual acceptable characters or ranges of characters. For example, if you wanted to match *LETTER0.DOC* through *LETTER9.DOC*, you could use this command:

```
[c:\] dir letter[0-9].doc
```

You could find all files that have a vowel as the second letter in their name this way. This example also demonstrates how to mix the wildcard characters:

```
[c:\] dir ?[aeiouy]*.*
```

You can exclude a group of characters or a range of characters by using an exclamation mark [**!**] as the first character inside the brackets. This example displays all filenames that are at least 2 characters long **except** those which have a vowel as the second letter in their names:

```
[c:\] dir ?[!aeiouy]*.*
```

The next example, which selects files such as *AIP*, *BIP*, and *TIP* but not *NIP*, demonstrates how you can use multiple ranges inside the brackets. It will accept a file that begins with an **A**, **B**, **C**, **D**, **T**, **U**, or **V**:

```
[c:\] dir [a-dt-v]ip
```

You may use a question mark character inside the brackets, but its meaning is slightly different than a normal (unbracketed) question mark wildcard. A normal question mark wildcard matches any character, but will be ignored when matching a name shorter than the wildcard specification, as described above. A question mark inside brackets will match any character, but will **not** be discarded when matching shorter filenames. For example:

```
[c:\] dir letter[?].doc
```

will display *LETTER1.DOC* and *LETTERA.DOC*, but not *LETTER.DOC.*

A pair of brackets with no characters between them **[]**, or an exclamation point and question mark together **[!?]**,will match only if there is no character in that position. For example,

```
[c:\] dir letter[].doc
```

will not display *LETTER1.DOC* or *LETTERA.DOC*, but will display *LETTER.DOC*. This is most useful for commands like

```
[c:\] dir /I"[]" *.btm
```

which will display a list of all *.BTM* files which **don't** have a description, because the empty brackets match only an empty description string (DIR /I selects files to display based on their descriptions).

You can repeat any of the wildcard characters in any combination you desire within a single file name. For example, the following command lists all files which have an **A**, **B**, or **C** as the third character, followed by zero or more additional characters, followed by a **D**, **E**, or **F**, followed optionally by some additional characters, and with an extension beginning with **P** or **Q**. You probably won't need to do anything this complex, but we've included it to show you the flexibility of extended wildcards:

```
[c:\] dir ??[abc]*[def]*.[pq]*
```

You can also use the square bracket wildcard syntax to work around a conflict between long filenames containing semicolons [**;**], and the use of a semicolon to indicate an include list. For example, if you have a file named *C:\DATA\LETTER1;V2* and you enter this command:

```
      [c:\] del \data\letter1;v2
```

you will not get the results you expect. Instead of deleting the named file, 4NT will attempt to delete *LETTER1* and then *V2*, because the semicolon indicates an include list. However if you use square brackets around the semicolon it will be interpreted as a filename character, and not as an include list separator. For example, this command would delete the file named above:

```
      [c:\] del \data\letter1[;]v2
```

Extra caution should be taken using wildcards on long file names because operations using wildcards will be performed on both long and short filenames. See LFN File Searches for additional details.

Date, Time, and Size Ranges

Most internal commands which accept wildcards also allow date, time, and size ranges to further define the files that you wish to work with. 4NT will examine each file's size and timestamp (a record of when the file was created, last modified, or last accessed) to determine if the file meets the range criteria you have specified.

(4NT also supports File Exclusion Ranges to exclude files from a command. These are similar to date, time, and size ranges, but have a slightly different purpose and therefore are documented separately.)

A range begins with the switch character (**/**), followed by a left square bracket ("**[**") and a character that specifies the range type: "s" for a size range, "d" for a date range, or "t" for a time range. The "s", "d", or "t" is followed by a start value, and an optional comma and end value. The range ends with a right square bracket ("**]**"). For example, to select files between 100 and 200 bytes long you could use the range **/[s100,200]**.

All ranges are inclusive. For example, a size range which selects files from 10,000 to 20,000 bytes long will match files that are exactly 10,000 bytes and 20,000 bytes long, as well as all sizes in between; a date range that selects files last modified between 10-27-97 and 10-30-97 will include files modified on each of those dates, and on the two days in between.

If you reverse range start and end values the command processor will recognize the reversal, and will use the second (lower) value as the start point of the range and the first (higher) value as its end point. For example, the range above for files between 100 and 200 bytes long could also be entered as **/[s200,100]**.

See the individual range types for details on specifying ranges:

Date Ranges

Time Ranges

Size Ranges

Using Ranges

If you combine two types of ranges, a file must satisfy both ranges to be included. For example, **/[d2-8-97,2-9-97] /[s1024,2048]** means files last modified on February 8 or February 9, 1997, which are also between 1,024 and 2,048 bytes long.

When you use a date, time, or size range in a command, it should immediately follow the command name. Unlike some command switches which apply to only part of the command line, the range usually applies to all file names specified for the command. Any exceptions are noted in the descriptions of individual commands.

For example, to get a directory of all the **.C* files dated October 1, 1997, you could use this command:

```
[c:\] dir /[d10-1-97,+0] *.c
```

To delete all of the 0-byte files on your hard disk, you could use this command:

```
[c:\] del /[s0,0] *.* /s
```

And to copy all of the non-zero byte files that you changed yesterday or today to your floppy disk, you can use this command:

```
[c:\] copy /[d-1] /[s1] *.* a:
```

It can be complex to type all of the elements of a range, especially when it involves multiple dates and times. In this case you may find it easier to use aliases for common operations. For example, if you often wish to select from *.DAT* files modified over the last three days and copy the selected files to the floppy disk, you might define an alias like this:

```
alias workback `select /[d-2] copy (*.dat) a:`
```

For more complex requirements, you may want to use internal variables (*e.g.* _DATE or _TIME and variable functions (*e.g.* @DATE, @TIME, @MAKEDATE, @MAKETIME, @FILEDATE, @FILETIME, or @EVAL). These variables and functions allow you to perform arithmetic and date / time calculations.

File systems which support long filenames maintain 3 sets of dates and times for each file: creation, last access, and last write. By default, date and time ranges work with the last write time stamp. You can use the "last access" (a) or "created" (c) time stamp in a date or time range with the syntax:

```
/[da...]  or  /[dc...]  or .. /[ta...]  or  /[tc...]
```

For example, to select files that were last accessed yesterday or today:

```
/[da-1]
```

The LFN file system used by Windows NT on FAT volumes, stores an incomplete time stamp for last access. You can select files by the date of last access, but not by the time of last access, since the time of last access is not retained by the operating system.

Date, time, and size ranges can be used with the ATTRIB, COPY, DEL, DESCRIBE, DIR, EXCEPT, FFIND, FOR, LIST, MOVE, RD, REN, SELECT, and TYPE commands. They cannot be used with filename completion or in filename arguments for variable functions.

Date Ranges

Date ranges select files that were created or last modified at any time between the two dates. For example, **/[d12-1-97,12-5-97]** selects files that were last modified between December 1, 1997, and December 5, 1997.

The time for the starting date defaults to 00:00:00 and the time for the ending date defaults to 23:59:59. You can alter these defaults, if you wish, by including a start and stop time inside the date range. The time is separated from the date with an at sign [**@**]. For example, the range **/[d7-1-97@8:00a,7-3-97@6:00p]** selects files that were modified at any time between 8:00 am on July 1, 1997 and 6:00 PM on July 3, 1997. If you prefer, you can specify the times in 24-hour format (*e.g.*, @18:00 for the end time in the previous example).

If you omit the second argument in a date range, 4NT substitutes the current date and time. For example, **/[d10-1-97]** selects files dated between October 1, 1997 and today.

You can use an offset value for either the beginning or ending date, or both. An offset begins with a plus sign [**+**] or a minus sign [**-**] followed by an integer. If you use an offset for the second value, it is calculated relative to the first. If you use an offset for the first (or only) value, the current date is used as the basis for calculation. For example:

```
Specification    Selects Files

/[d10-27-97,+3]  modified between 10-27-97 and 10-30-97
/[d10-27-97,-3]  modified between 10-24-97 and 10-27-97
/[d-0]           modified today (from today minus zero days, to today)
/[d-1]           modified yesterday or today (from today minus one day,
                 to today)
/[d-1,+0]        modified yesterday (from today minus one day, to zero
                 days after that)
```

As a shorthand way of specifying files modified today, you can also use **/[d]**; this has the same effect as the **/[d-0]** example shown above.

To select files last modified **n** days ago or earlier, use **/[d-n,1/1/80]**. For example, to get a directory of all files last modified 3 days or more before today (*i.e.*, those files **not** modified within the last 3 days), you could use this command:

```
c:\> dir /[d-3,1/1/80]
```

This reversed date range (with the later date given first) will be handled correctly by 4NT. It takes advantage of the facts that an offset in the **start** date is relative to today, and that the base or "zero" point for PC file dates is January 1, 1980.

You cannot use offsets in the time portion of a date range (the part after an at sign), but you can combine a time with a date offset. For example, **/[d12-8-97@12:00,+2@12:00]** selects files that were last modified between noon on December 8 and noon on December 10, 1997. Similarly, **/[d-2@15:00,+1]** selects files last modified between 3:00 PM the day before yesterday and the end of the day one day after that, *i.e.*, yesterday. The second time defaults to the end of the day because no time is given.

File systems which support long filenames maintain 3 sets of dates and times for each file: creation, last access, and last write. By default, date ranges work with the last write date/time stamp. You can use the "last access" (a) or "created" (c) date/time stamp in a date range with the syntax:

```
/[da...]  or  /[dc...]
```

For example, to select files that were last accessed yesterday or today:

```
/[da-1]
```

Time Ranges

A time range specifies a file modification time without reference to the date. For example, to select files modified between noon and 2:00 PM on any date, use **/[t12:00p,2:00p]**. The times in a time range can either be in 12-hour format, with a trailing "a" for AM or "p" for PM, or in 24-hour format.

If you omit the second argument in a time range, you will select files that were modified between the first time and the current time, on any date. You can also use offsets, beginning with a plus sign [**+**] or a minus sign [**-**] for either or both of the arguments in a time range. The offset values are interpreted as minutes. Some examples:

```
Specification    Selects Files

/[t12:00p,+120]  modified between noon and 2:00 PM on any date
/[t-120,+120]    modified between two hours ago and the current time on
                 any date
/[t0:00,11:59]   modified in the morning on any date
```

File systems which support long filenames maintain 3 sets of dates and times for each file: creation, last access, and last write. By default, time ranges work with the last write time stamp. You can use the "last access" (a) or "created" (c) time stamp in a time range with the syntax:

```
/[ta...]  or  /[tc...]
```

The LFN file system used by Windows NT on FAT volumes, stores an incomplete time stamp for last access. You can select files by the date of last access, but not by the time of last access, since the time of last access is not retained by the operating system.

Size Ranges

Size ranges simply select files whose size is between the limits given. For example, **/[s10000,20000]** selects files between 10,000 and 20,000 bytes long.

Either or both values in a size range can end with "k" to indicate thousands of bytes, "K" to indicate kilobytes (1,024 bytes), "m" to indicate millions of bytes, or "M" to indicate megabytes (1,048,576 bytes). For example, the range above could be rewritten as **/[s10k,20k]**.

All ranges are inclusive. Both examples above will match files that are exactly 10,000 bytes and 20,000 bytes long, as well as all sizes in between.

The second argument of a size range is optional. If you use a single argument, like **/[s10k]**, you will select files of that size or larger. You can also precede the second argument with a plus sign [**+**]; when you do, it is added to the first value to determine the largest file size to include in the search. For example, **/[s10k,+1k]** select files from 10,000 through 11,000 bytes in size.

Some further examples of size ranges:

```
Specification  Selects Files

/[s0,0]        of length zero(empty)
/[s1M]         1 megabyte or more in length
/[s10k,+200]   between 10,000 and 10,200 bytes
```

File Exclusion Ranges

Most internal commands which accept wildcards also accept file exclusion ranges to further define the files that you wish to work with.   4NT examines each file name and excludes files that match the names you have specified in a file exclusion range.

A file exclusion range begins with the switch character (usually a slash), followed by a left square bracket and an exclamation mark ("**[!**")   The range ends with a right square bracket ("**]**").

Inside the brackets, you can list one or more filenames to be excluded from the command.   The filenames can include wildcards and extended wildcards, but cannot include path names or drive letters.

The following example will display all files in the current directory except backup files (files with the extension *.BAK* or *.BKP*):

```
[c:\] dir /[!*.bak *.bkp] *.*
```

You can combine file exclusion ranges with date, time, and size ranges.   This example displays all files that are 10K bytes or larger in size and that were created in the last 7 days, except *.C* and *.H* files:

```
[c:\] dir /[s10k] /[d-7] /[!*.c *.h] *.*
```

File exclusion ranges will only work for 4NT internal commands.   The EXCEPT command can be used to exclude files from processing by many external commands.

Multiple Filenames

Most file processing commands can work with multiple files at one time. To use multiple file names, you simply list the files one after another on the command line, separated by spaces. You can use wildcards in any or all of the filenames. For example, to copy all *.TXT* and *.DOC* files from the current directory to drive A, you could use this command:

```
[c:\] copy *.txt *.doc a:
```

If the files you want to work with are not in the default directory, you must include the full path with each filename:

```
[c:\] copy a:\details\file1.txt a:\details\file1.doc c:
```

Multiple filenames are handy when you want to work with a group of files which cannot be defined with a single filename and wildcards. They let you be very specific about which files you want to work with in a command.

When you use multiple filenames with a command that expects both a source and a destination, like COPY or MOVE, be sure that you always include a specific destination on the command line. If you don't, the command will assume that the last filename is the destination and may overwrite important files.

Like extended wildcards and include lists, the multiple filenames will work with internal commands but not with external programs, unless those programs have been written to handle multiple file names on the command line.

If you have a list of files to process that's too long to put on the command line or too time-consuming to type, see FOR or SELECT for other ways of passing multiple file names to a command.

Include Lists

Any internal command that accepts multiple filenames will also accept one or more include lists. An include list is simply a group of filenames, with or without wildcards, separated by semicolons [;]. All files in the include list must be in the same directory. You may not add a space on either side of the semicolon.

For example, you can shorten this command which uses multiple file names:

```
[c:\] copy a:\details\file1.txt a:\details\file1.doc c:
```

to this using an include list:

```
[c:\] copy a:\details\file1.txt;file1.doc c:
```

Include lists are similar to multiple filenames, but have three important differences. First, you don't have to repeat the path to your files if you use an include list, because all of the included files must be in the same directory. Second, if you use include lists, you aren't as likely to accidentally overwrite files if you forget a destination path for commands like COPY, because the last name in the list will be part of the include list, and won't be seen as the destination file name. (Include lists can only be used as the *source* parameter – the location files are coming from – for COPY and other similar commands. They cannot be used to specify a destination for files.)

Third, multiple filenames and include lists are processed differently by the DIR and SELECT commands. If you use multiple filenames, all of the files matching the first filename are processed, then all of the files matching the second name, and so on. When you use an include list, all files that match any entry in the include list are processed together, and will appear together in the directory display or SELECT list. You can see this difference clearly if you experiment with both techniques and the DIR command. For example:

```
[c:\] dir *.txt *.doc
```

will list all the *.TXT* files with a directory header, the file list, and a summary of the total number of files and bytes used. Then it will do the same for the *.DOC* files. However,

```
[c:\] dir *.txt;*.doc
```

will display all the files in one list.

Like extended wildcards and multiple filenames, the include list feature will work with internal commands, but not with external programs (unless they have been programmed especially to support them). The maximum length of an include list is 260 characters.

LFN File Searches

Under Windows NT (version 3.5 and later), files on VFAT volumes can have both a long file name (LFN) and a short FAT-compatible file name. 4NT normally examines both forms of each file name when searching for files. It does so in order to remain compatible with the default command processor, *CMD.EXE*.

The long filename is checked first, and if it does not match then the short name is checked. Matching files which have only a short filename will be found during the first search, because in that case Windows NT treats the short name as if it were a long name.

For example, suppose you have two files in a directory with these names:

Long Name	Short Name
Letter Home.DOC	LETTER~1.DOC
Letter02.DOC	LETTER02.DOC

A search for *LETTER??.DOC* will find both files. The second file (*LETTER02.DOC*) will be found during the search of long filenames. The first file (*Letter Home.DOC*) will be found during the search of short filenames.

Take extra care when you use wildcards to perform operations on LFN volumes because you may select more files than you intended. For example, Windows NT often creates short filenames that end "~1.", "~2.", etc. If you use a command like:

```
del *1.*
```

you will delete all such files, including most files with long filenames, which is probably **not** the result you intended!

Executable Extensions

The syntax for creating an executable extension is:

```
set .ext=command [options]
```

This tells 4NT to run the specified command whenever you name a file with the extension **.ext** at the prompt.

.EXT is the executable file extension; *command* is the name of the internal command, external program, alias, or batch file to run; and *[options]* are any command-line startup options you want to specify for the program, batch file, or alias.

Normally, when you type a filename (as opposed to an alias or internal command name) as the first word on the command line, 4NT looks for a file with that name to execute. The file's extension may be *.EXE* or *.COM* to indicate that it contains a program, it may have a batch file extension like *.BTM*, or the file's contents may indicate that it is executable.

You can add to the default list of extensions, and have 4NT take the action you want with files that are not executable programs or batch files. The action taken is always based on the file's extension. For example, you could start your text editor whenever you type the name of a *.DOC* file, or start your database manager whenever you type the name of a *.DAT* file.

Windows also includes the ability to associate file extensions with specific applications. See Using Windows File Associations for details on file associations and their relationship to 4NT executable extensions.

You can use environment variables to define the internal command, external program, batch file, or alias to run for each defined file extension. To create an executable extension, use the SET command to create a new environment variable. An environment variable is recognized as an executable extension if its name begins with a period.

For example, if you want to run a word processor called *EDITOR* whenever you type the name of a file that has an extension of *.EDT*, you could use this command:

```
[c:\] set .edt=c:\edit\editor.exe
```

If the command specified in an executable extension is a batch file or external program, 4NT will search the PATH for it if necessary. However, you can make sure that the correct program or batch file is used, and speed up the executable extension, by specifying the full name including drive, path, filename, and extension.

Once an executable extension is defined, any time you name a file with that extension the corresponding program, batch file, or alias is started, with the name of your file passed to it as a parameter.

The following example defines *B.EXE* (the Boxer text editor) as the processor for *.MAK* files:

```
[c:\] set .mak=c:\boxer\b.exe -s
```

With this definition, if you have a file named *INIT.MAK* in the current directory and enter the command:

```
[c:\source] init
```

4NT will execute the command:

```
      c:\boxer\b.exe -s c:\source\init.mak
```

Notice that the full pathname of *INIT.MAK* is automatically included. If you enter parameters on the command line, they are appended to the end of the command. For example, if you changed the above entry to:

```
      [c:\source] init -w
```

the command processor would execute the command:

```
      c:\boxer\b.exe -s c:\source\init.mak -w
```

In order for executable extensions to work, the command, program, batch file, or alias must be able to interpret the command line properly. For example, if a program you want to run doesn't accept a file name on its command line as shown in these examples, then executable extensions won't work with that program.

Executable extensions may include wildcards, so you could, for example, run your text editor for any file with an extension beginning with **T** by defining an executable extension called *.T**. Extended wildcards (*e.g.*, **DO[CT]** for *.DOC* and *.DOT* files) may also be used.

The search for executable files starts in the current directory, then proceeds to each subdirectory specified by the PATH environment variable (if a "**.**" is used in the PATH the current directory is **not** searched first; see the PATH command for details).

You may need to take this search order into account when using executable extensions. Using the *MAK* example above, if you had a file named *SORT.MAK* in the current directory and entered the command SORT TESTFILE.TXT, your command would run the Boxer editor specified by the executable extension, instead of finding the standard SORT command as you perhaps intended. You can get around this by remembering that the SORT command is in the file *SORT.EXE*. If you entered the command SORT.EXE TESTFILE.TXT then the *.MAK* executable extension would not be checked, and the search would continue until it found the *SORT.EXE* file.

To remove an executable extension, use the UNSET command to remove the corresponding variable.

Using Windows File Associations

Windows NT includes the ability to associate file extensions with specific applications; this feature is sometimes called "file associations".   For example, within Windows NT a graphics program might be associated with files with a *.BMP* extension, while Notepad could be associated with files with a *.TXT* extension.   Windows NT supports two different kinds of file associations ("direct" and "indirect"); for a complete description see Windows File

When you attempt to start an application from the command line or a batch file, 4NT first searches for an external program file with a standard extension (*.COM*, *.EXE*, etc.).   It then checks 4NT executable extensions, followed by direct file associations inherited from Windows NT.   If   all of these tests fail, 4NT passes the command name to Windows NT to see if it can find an indirect association.

4NT offers two commands which provide limited control over indirect file associations.   Both should be used with caution to avoid creating errors in the registry or damaging existing file types.   The ASSOC command modifies or displays the associations between extensions and file types in the Windows registry.   The FTYPE command modifies or displays the command used to open a file of a specified type.

Executable extensions defined in 4NT always take precedence over the direct and indirect file associations defined in Windows NT.   For example, if you associate the *.TXT* extension with your own editor using a 4NT executable extension, and Windows NT has associated *.TXT* with Notepad, your setting will have priority, and the association with Notepad will be ignored when you invoke a *.TXT* file from within 4NT.

Unfortunately, it is not unusual to find both a direct association and an indirect association in the Windows registry for the same extension.   This can happen when an ill-behaved install or uninstall program modifies the wrong registry entry, or when a 16-bit application registers one type of association and a 32-bit application registers the other type for the same extension.   For example, under Windows 95 you might have a direct association between *.GIF* files and a 16-bit graphics program, and an indirect association between *.GIF* files and a newer 32-bit application.   When this happens 4NT will find the direct association first, which may not be the result you want.

To address such problems, you can correct the registry entries (**use extreme caution** when modifying the registry manually as errors in the registry can prevent your system from booting); create a 4NT executable extension which explicitly specifies the application to run; disable the loading of direct associations from the Startup page of the OPTION dialogs, or with a LoadAssociations = No directive in the *4NT.INI* file; or disable an individual association with the UNSET command.

To disable individual direct file associations while you are working in 4NT, use the UNSET command plus the appropriate file extension for each association that you want 4NT to ignore.   UNSET will disable that file association within 4NT, but will not affect the use of the association by other Windows applications. For example, to disable a direct association between *.WAV* files and a sound player while you are working in 4NT, you could use this command:

```
        [c:\] unset .wav
```

This approach can only be used to disable direct associations.   Indirect associations cannot be disabled (although they can be overridden with a 4NT executable extension).

Using Internet URLs

If you type an Internet URL (Uniform Resource Locator) which begins with **http:** at the prompt, 4NT will pass the URL to Windows NT.   Normally Windows NT will start your web browser, and request that the browser retrieve the page pointed to by the URL.   This feature will only work if Windows NT can find the proper association between the **http:** prefix and the browser software.   While this association is standard for most browser installations, it may not be present on all systems.

The ability to "start" URLs in this way is restricted to those beginning with **http:**.   Other standard prefixes such as **ftp:**, **mail:**, and **news:** cannot be started directly from the prompt; you must enter these URLs directly into the browser software.

See Waiting for Applications to Finish for information on problems with waiting for the browser to finish after starting a URL.

Waiting for Applications to Finish

When you start a Windows application from the prompt, 4NT does not normally wait for the application to finish before returning to the prompt. This default behavior allows you to continue your work at the prompt while the application is running. You can force 4NT to wait for applications to finish before continuing by selecting the "Wait for Completion" option on the Options 2 page of the OPTION dialogs, with the ExecWait directive in the *.INI* file, or with the START command's **/WAIT** switch (you can also use START to control many other aspects of how your applications are started).

Regardless of the ExecWait setting, 4NT always waits for applications which are run from batch files before continuing with subsequent commands in the batch file. To start an application from a batch file and continue with the batch file immediately, without waiting for the application to finish, use the START command (without the **/WAIT** switch).

Due to the way Windows NT handles URLs, you cannot wait for the browser software to finish when you enter an **http:** URL at the prompt; in this situation, 4NT always displays the next prompt immediately. See Using Internet URLs for information about starting your browser from the prompt.

Directory Navigation

The operating system and command processor remember both a **current** or **default drive** for your system as a whole, and a **current** or **default directory** for every drive in your system.   The current directory on the current drive is sometimes called the **current working directory**.

With traditional command processors, you change the current drive by typing the new drive letter plus a colon at the prompt, and you change the current working directory with the CD command. 4NT supports those standard features, and offer a number of enhancements to make directory navigation much simpler and faster.

The 4NT directory navigation features are in three groups: features which help the command processor find the directory you want, methods for initiating a directory change with a minimal amount of typing, and methods for returning easily to directories you've recently used.   Each group is summarized below.

Finding Directories

Traditional command processors require you to explicitly type the name of the directory you want to change to. 4NT supports this method, and also offers two significant enhancements:

- » **Extended Directory Searches** allow the command processor to search a "database" of all the directories on your system to find the one you want.
- » The **CDPATH** allows you to enter a specific list of directories to be searched, rather than searching a database.   Use CDPATH instead of Extended Directory Searches if you find the extended searches too broad, or your hard drive has too many directories for an efficient search.

Initiating a Directory Change

4NT supports the traditional methods of changing directories, and also offers several more flexible approaches:

- » **Automatic directory changes** allow you to type a directory name at the prompt and switch to it automatically, without typing an explicit CD or similar command.
- » The **CD command** can change directories on a single drive, and can return to the most recently used directory.
- » The **CDD command** changes drive and directory at the same time, and can return to the most recently used drive and directory.
- » The **PUSHD command** changes the drive and directory like CDD, and records the previous directory in a directory "stack."   You can view the stack with DIRS and return to the directory on the top of the stack with POPD.

CDD, PUSHD, and automatic directory changes can also change to a network drive and directory mapped to a drive letter or specified with a UNC name.

Returning to a Previous Directory

Traditional command processors do not remember previously-used directories, and can only "return" to a directory by changing back to it with a standard drive change or CD command. 4NT supports three additional methods for returning to a previous directory:

- » The **CD -** and **CDD - commands** can be used to return to the previous working directory (the one you used immediately before the current directory). Use these commands if you are working in two directories and alternating between them.

» The **directory history window** allows you to select one of several recently-used directories from a popup list and return to it immediately. The window displays the contents of the directory history list.

» The **POPD command** will return to the last directory saved by PUSHD. The directory stack holds 511 characters, enough for 20 to 40 typical drive and directory entries.

Extended Directory Searches

When you change directories with an automatic directory change, CD, CDD, or PUSHD command, 4NT must find the directory you want to change to. To do so, the command processor first uses the traditional method to find a new directory: it checks to see whether you have specified either the name of an existing subdirectory below the current directory, or the name of an existing directory with a full path or a drive letter. If you have, the command processor changes to that directory, and does no further searching.

This traditional search method requires that you navigate manually through the directory tree, and type the entire name of each directory you want to change to. Extended Directory Searches speed up the navigation process dramatically by allowing the command processor to find the directory you want, even if you only enter a small part of its name.

When the traditional search method fails, 4NT tries to find the directory you requested via the CDPATH, then via an Extended Directory Search. This section covers only Extended Directory Searches, which are more flexible and more commonly used than CDPATH.

Extended Directory Searches use a database of directory names to facilitate changing to the correct directory. The database is used only if Extended Directory Searches are enabled, and if the explicit directory search and CDPATH search fail to find the directory you requested.

An extended directory search automatically finds the correct path to the requested directory and changes to it if that directory exists in your directory database. If more than one directory in the database matches the name you have typed, a popup window appears and you can choose the directory you want.

You can control the color, position and size of the popup directory search window from the Command line 2 page of the OPTION dialogs, or with directives in the *.INI* file, including CDDWinLeft, CDDWinTop, CDDWinWidth, and CDDWinHeight, and CDDWinColors). You can also change the keys used in the popup window with key mapping directives in the *.INI* file.

To use extended directory searches, you must explicitly enable them (see below) and also create the directory database.

The Extended Search Database

To create or update the database of directory names, use the **CDD /S** command. When you create the database with CDD /S, you can specify which drives should be included. If you enable Extended Directory Searches and do not create the database, it will be created automatically the first time it is required, and will include all local hard drives.

The database is stored in the file *JPSTREE.IDX*, which is placed in the root directory of drive C: by default. The same tree file is used by all JP Software command processors. You can specify a different location for this file on the Command Line 2 page of the OPTION dialogs, or with the TreePath *.INI* directive. If you are using 2 or more of our products on your computer and want to have different drives stored in the database for each, use the dialogs or the TreePath directive to place their database directories in different locations.

If you use an internal 4NT command to create or delete a directory, the directory database is automatically updated to reflect the change to your directory structure. The updates occur if the command processor can find the *JPSTREE.IDX* file in the root directory of drive C: or in the location specified by the TreePath *.INI* directive.

The internal commands which can modify the directory structure and cause automatic updates of the file are MD, RD, COPY /S, DEL /X, MOVE /S, and REN. The **MD /N** command can be used to create a directory without updating the directory database. This is useful when creating a temporary directory which you do not want to appear in the database.

Enabling Extended Searches

To enable extended directory searches and control their operation, you must set the **FuzzyCD** directive in the *.INI* file. You can set FuzzyCD with the Search Level option on the Command Line 2 page of the OPTION dialogs or by editing the *.INI* file manually.

> If **FuzzyCD = 0**, extended searches are disabled, the *JPSTREE* database is ignored, and **CD**, **CDD**, **PUSHD**, and automatic directory changes search for directories using only explicit names and CDPATH. This is the default.
>
> If **FuzzyCD = 1** and an extended search is required, then the command processor will search the *JPSTREE* database for directory names which **exactly match** the name you specified.
>
> If **FuzzyCD = 2** and an extended search is required, the command processor will search the database for exact matches first, just as when FuzzyCD = 1. If the requested directory is not found, it will search the database a second time looking for directory names that **begin with** the name you specified.
>
> If **FuzzyCD = 3** and an extended search is required, the command processor will search the database for exact matches first, just as when FuzzyCD = 1. If the requested directory is not found, it will search the database a second time looking for directory names that **contain** the name you specified anywhere within them.

For example, suppose that you have a directory called *C:\DATA\MYDIR*, CDPATH is not set, and *C:\DATA* is **not** the current directory on drive C:. The following chart shows what CDD command you might use to change to this directory.

Fuzzy CD	CDD Command
0	`cdd c:\data\mydir`
1	`cdd mydir`
2	`cdd myd`
3	`cdd yd`

An extended directory search is not used if you specify a full directory path (one beginning with a backslash [\], or a drive letter and a backslash). If you use a name which begins with a drive letter (*e.g. C:MYDIR*), the extended search will examine only directories on that drive.

Forcing an Extended Search with Wildcards

Normally you type a specific directory name for the command processor to locate, and the search proceeds as described in the preceding sections. However, you can also force the command processor to perform an extended directory search by using wildcard characters in the directory name. If you use a wildcard, an extended search will occur whether or not extended searches have been enabled.

When 4NT is changing directories and it finds wildcards in the directory name, it skips the explicit search and CDPATH steps and goes directly to the extended search.

If a single match is found, the change is made immediately. If more than one match is found, a popup window is displayed with all matching directories.

Wildcards can only be used in the final directory name in the path (after the last backslash in the path name). For example you can find *COMM*A*.** (all directories whose parent directory is COMM and which have an A somewhere in their names), but you cannot find *CO?M*A*.** because it uses a wildcard before the last backslash.

If you use wildcards in the directory name as described here, and the extended directory search database does not exist, it will be built automatically the first time a wildcard is used.   You can update the database at any time with CDD /S.

Internally, extended directory searches use wildcards to scan the directory database.   If FuzzyCD is set to 2, an extended search looks for the name you typed followed by an asterisk (*i.e. DIRNAME**).   If FuzzyCD is set to 3, it looks for the name preceded and followed by an asterisk (*i.e. *DIRNAME*)*.

These internal wildcards will be used in addition to any wildcards you use in the name.   For example if you search for *ABC?DEF* (ABC followed by any character followed by DEF) and FuzzyCD is set to 3, the command processor will actually search the directory database for **ABC?DEF**.

CDPATH

When you change directories with an automatic directory change, CD, CDD, or PUSHD command, 4NT must find the directory you want to change to. To do so, the command processor first uses the traditional method to find a new directory.

When the traditional search method fails, 4NT tries to find the directory you requested via the CDPATH, then via an Extended Directory Search. This section covers only CDPATH.

Enabling both CDPATH and Extended Directory Searches can yield confusing results, so we recommend that you do not use both features at the same time. If you prefer to explicitly list where the command processor should look for directories, use CDPATH. If you prefer to have the command processor look at all of the directory names on your disk, use Extended Directory Searches.

CDPATH is an environment variable, and is similar to the PATH variable used to search for executable files: it contains an explicit list of directories to search when attempting to find a new directory. The command processor appends the specified directory name to each directory in CDPATH and attempts to change to that drive and directory. It stops when it finds a match or when it reaches the end of the CDPATH list.

CDPATH is ignored if a complete directory name (one beginning with a backslash [\]) is specified, or if a drive letter is included in the name. It is only used when a name is given with no drive letter or leading backslash.

CDPATH provides a quick way to find commonly used subdirectories in an explicit list of locations. You can create CDPATH with the SET command. The format of CDPATH is similar to that of PATH: a list of directories separated by semicolons [**;**]. For example, if you want the directory change commands to search the C:\DATA directory, the D:\SOFTWARE directory, and the root directory of drive E:\ for the subdirectories that you name, you should create CDPATH with this command:

```
[c:\] set cdpath=c:\data;d:\software;e:\
```

Suppose you are currently in the directory C:\WP\LETTERS\JANUARY, and you'd like to change to D:\SOFTWARE\UTIL. You could change directories explicitly with the command:

```
[c:\wp\letters\january] cdd d:\software\util
```

However, because the *D:\SOFTWARE* directory is listed in your CDPATH variable as shown in the previous example (we'll assume it is the first directory in the list with a UTIL subdirectory), you can simply enter the command:

```
[c:\wp\letters\january] cdd util
```

or, using an automatic directory change:

```
[c:\wp\letters\january] util\
```

to change to D:\SOFTWARE\UTIL.

As it handles this request, the command processor looks first in the current directory, and attempts to find the *C:\WP\LETTERS\JANUARY\UTIL* subdirectory. Then it looks at CDPATH, and appends the name you entered, *UTIL*, to each entry in the CDPATH variable — in other words, it tries to change to *C:\DATA\UTIL*, then to *D:\SOFTWARE\UTIL*. Because this change succeeds, the search stops and the directory change is complete.

Note: **_CDPATH** can be used as an alternative to CDPATH if you are using Microsoft Bookshelf, which uses a CDPATH variable for its own purposes.

Other Features

Page and File Prompts
Redirection and Piping
Critical Errors
Conditional Commands
Command Grouping
Escape Character

Page and File Prompts

Several 4NT commands can generate prompts, which wait for you to press a key to view a new page or to perform a file activity.

When 4NT is displaying information in page mode, for example with a DIR /P or SET /P command, it displays the message

```
Press Esc to Quit or any other key to continue...
```

At this prompt, you can press **Esc**, **Ctrl-C**, or **Ctrl- Break** if you want to quit the command.   You can press almost any other key to continue with the command and see the next page of information.

During file processing, if you have activated prompting with a command like DEL /P, you will see this prompt before processing every file:

```
Y/N/R ?
```

You can answer this prompt by pressing **Y** for "**Y**es, process this file;"   **N** for "**N**o, do not process this file;" **R** for "process the **R**emainder of the files without further prompting."   You can also press **Ctrl-C**, **Ctrl-Break**, or **Esc** at this prompt to cancel the remainder of the command.

Redirection and Piping

This section covers **redirection** and **piping**.   You can use these features to change how 4NT and some application programs handle input and output.

Internal commands and many external programs get their input from the computer's **standard input** device and send their output to the **standard output** device.   Some programs also send special messages to the **standard error** device.   Normally, the keyboard is used for standard input and the video screen for both standard output and standard error.

Redirection and piping allow you to change these assignments temporarily.

Redirection changes the standard input, standard output, or standard error device for a program or command from the default device (the keyboard or screen), to another device or to a file.

Piping changes the standard output and / or standard error device so that the output of one command becomes the standard input for another program or command.

Redirection

Redirection can be used to reassign the **standard input**, **standard output**, and **standard error** devices from their default settings (the keyboard and scree) to another device like the printer or serial port, to a file, or to the clipboard. You must use some discretion when you use redirection with a device; there is no way to get input from the printer, for example.

Redirection always applies to a specific command, and lasts only for the duration of that command. When the command is finished, the assignments for standard input, standard output, and standard error revert to whatever they were before the command.

In the descriptions below, **filename** means either the name of a file or of an appropriate device (PRN, LPT1, LPT2, or LPT3 for printers; COM1 to COM4 for serial ports; CON for the keyboard and screen; CLIP: for the clipboard; NUL for the "null" device, etc.).

Here are the standard redirection options supported by 4NT (see below for additional redirection options using numeric file handles):

< filename	To get input from a file or device instead of from the keyboard
> filename	Redirect standard output to a file or device
>& filename	Redirect standard output and standard error to a file or device
>&> filename	Redirect standard error only to a file or device

If you want to append output to the end of an existing file, rather than creating a new file, replace the first "**>**" in the last three commands above with "**>>**" (*i.e.*, use **>>**, **>>&**, and **>>&>**).

To use redirection, place the redirection symbol and filename at the end of the command line, after the command name and any parameters. For example, to redirect the output of the DIR command to a file called *DIRLIST*, you could use a command line like this:

```
[c:\] dir /b *.dat > dirlist
```

You can use both input and output redirection for the same command, if both are appropriate. For example, this command sends input to SORT from the file *DIRLIST*, and sends output from SORT to the file *DIRLIST.SRT*:

```
[c:\] sort < dirlist > dirlist.srt
```

You can redirect text to or from the Windows clipboard by using the pseudo-device name **CLIP:** (the colon is required).

If you redirect the output of a single internal command like DIR, the redirection ends automatically when that command is done. If you start a batch file with redirection, all of the batch file's output is redirected, and redirection ends when the batch file is done. Similarly, if you use redirection at the end of a command group, all of the output from the command group is redirected, and redirection ends when the command group is done.

When output is directed to a file with **>**, **>&**, or **>&>**, if the file already exists, it will be overwritten. You can protect existing files by using the SETDOS /N1 command, the "Protect redirected output files" setting on the Options 1 page of the OPTION dialogs, or the NoClobber directive in the *.INI* file.

When output is appended to a file with **>>**, **>>&**, or **>>&>**, the file will be created if it doesn't already exist. However, if NoClobber is set as described above, append redirection will not create a new file; instead, if the output file does not exist a "File not found" or similar error will be displayed.

You can temporarily override the current setting of NoClobber by using an exclamation mark [**!**] after the redirection symbol. For example, to redirect the output of DIR to the file *DIROUT*, and allow overwriting of any existing file despite the NoClobber setting:

```
[c:\] dir >! dirout
```

Redirection is fully nestable. For example, you can invoke a batch file and redirect all of its output to a file or device. Output redirection on a command within the batch file will take effect for that command only; when the command is completed, output will revert to the redirected output file or device in use for the batch file as a whole.

You can use redirection if you need to create a zero-byte file. To do so, enter **>filename** as a command, with no actual command before the **>** character.

In addition to the standard redirection options, 4NT also supports the Windows NT *CMD.EXE* syntax:

n>file	Redirect handle **n** to the named file
n>&m	Redirect handle **n** to the same place as handle **m**

[**n**] and [**m**] are one-digit file handles between 0 and 9. You may not put any spaces between the **n** and the **>**, or between the **>**, **&**, and **m** in the second form. Windows NT interprets "0" as standard input, "1" as standard output, and "2" as standard error. Handles 3 to 9 will probably not be useful unless you have an application which uses those handles for a specific, documented purpose, or you have opened a file with the %@FILEOPEN variable function and the file handle is between 3 and 9.

The **n>file** syntax redirects output from handle **n** to a file. You can use this form to redirect two handles to different places. For example:

```
[c:\] dir > outfile 2> errfile
```

sends normal output to a file called *OUTFILE* and any error messages to a file called *ERRFILE*.

The **n>&m** syntax redirects handle **n** to the same location as the previously assigned handle **m**. For example, to send standard error to the same file as standard output, you could use this command:

```
[c:\] dir > outfile 2>&1
```

Notice that you can perform the same operations by using standard 4NT redirection features. The two examples above could be written as

```
[c:\] dir > outfile >&> errfile
```

and

```
[c:\] dir >&outfile
```

Piping

You can create a "pipe" to send the standard output of one command to the standard input of another command:

command1 | command2	Send the standard output of *command1* to the standard input of *command2*
command1 |& command2	Send the standard output and standard error of *command1* to the standard input of *command2*

For example, to take the output of the SET command (which displays a list of your environment variables and their values) and pipe it to the SORT utility to generate a sorted list, you would use the command:

```
[c:\] set | sort
```

To do the same thing and then pipe the sorted list to the internal LIST command for full-screen viewing:

```
[c:\] set | sort | list
```

The TEE and Y commands are "pipe fittings" which add more flexibility to pipes.

Like redirection, pipes are fully nestable.   For example, you can invoke a batch file and send all of its output to another command with a pipe.   A pipe on a command within the batch file will take effect for that command only; when the command is completed, output will revert to the pipe in use for the batch file as a whole.

4NT implements pipes by starting a new process for the receiving program instead of using temporary files.   The sending and receiving programs run simultaneously; the sending program writes to the pipe and the receiving program reads from the pipe.   When the receiving program finishes reading and processing the piped data, it ends automatically.

When you use pipes with 4NT make sure you think about any possible consequences that can occur from using a separate process to run the receiving program.

Critical Errors

Windows NT watches for physical errors during input and output operations. Physical errors are those due to hardware problems, such as trying to read a floppy disk while the drive door is open.

These errors are called **critical errors** because Windows NT, 4NT, or your application program cannot proceed until the error is resolved.

When a critical error occurs, you will see a message asking you to choose one of four error handling options. The message comes from the Windows NT or 4NT, and will vary slightly depending on whether you are in full-screen or windowed mode. You can respond with a mouse click or menu selection. However, the options and their meanings are similar in all cases:

Retry	Retry the operation. Choose this option if you have corrected the problem.
Ignore	Ignore the error and continue. Use caution when choosing this option. Ignoring critical errors, especially on the hard disk, can cause errors in your applications or damage data on the disk.
Fail	Tell the program that the operation failed. This option returns an error code to the command processor or to the application program that was running when the error occurred. 4NT generally stops the current command when an operation fails. This option is not available for all errors; if you don't see it, use Abort instead.
Abort	Abort the program. Choose this option to stop the program that was running when the error occurred. Choosing Abort after an error in 4NT will abort the command, but not the command processor itself.

Conditional Commands

When an internal command or external program finishes, it returns a result called the exit code. Conditional commands allow you to perform tasks based upon the previous command's exit code. Many programs return a 0 if they are successful and a non-zero value if they encounter an error.

If you separate two commands by **&&** (AND), the second command will be executed only if the first returns an exit code of 0. For example, the following command will only erase files if the BACKUP operation succeeds:

```
[c:\] backup c:\ a: && del c:\*.bak;*.lst
```

If you separate two commands by **||** (OR), the second command will be executed only if the first returns a non-zero exit code. For example, if the following BACKUP operation fails, then ECHO will display a message:

```
[c:\] backup c:\ a: || echo Error in the backup!
```

All internal commands return an exit code, but not all external programs do. Conditional commands will behave unpredictably if you use them with external programs which do not return an explicit exit code. To determine whether a particular external program returns a meaningful exit code use an **ECHO %?** command immediately after the program is finished. If the program's documentation does not discuss exit codes you may need to experiment with a variety of conditions to see how the exit code changes.

Command Grouping

Command grouping allows you to logically group a set of commands together by enclosing them in parentheses. The parentheses are similar in function to the BEGIN and END block statements in some programming languages.

There are two primary uses for command grouping. One is to execute multiple commands in a place where normally only a single command is allowed. For example, suppose you want to execute two different REN commands in all subdirectories of your hard disk. You could do it like this:

```
[c:\] global ren *.wx1 *.wxo
[c:\] global ren *.tx1 *.txo
```

But with command grouping you can do the same thing in one command:

```
[c:\] global (ren *.wx1 *.wxo & ren *.tx1 *.txo)
```

The two REN commands enclosed in the parentheses appear to GLOBAL as if they were a single command, so both commands are executed for every directory, but the directories are only scanned once, not twice.

This kind of command grouping is most useful with the EXCEPT, FOR, GLOBAL, and IF commands. When you use this approach in a batch file you must either place all of the commands in the group on one line, or place the opening parenthesis at the end of a line and place the commands on subsequent lines. For example, the first two of these sequences will work properly, but the third will not:

```
for %f in (1 2 3) (echo hello %f & echo goodbye %f)
for %f in (1 2 3) (
echo hello %f
echo goodbye %f
)
for %f in (1 2 3) (echo hello %f
echo goodbye %f)
```

The second common use of command grouping is to redirect input or output for several commands without repeatedly using the redirection symbols. For example, consider the following batch file fragment which places some header lines (including today's date) and directory displays in an output file using redirection. The first ECHO command creates the file using **>**, and the other commands append to the file using **>>**:

```
echo Data files %_date > filelist
dir *.dat >> filelist
echo. >> filelist
echo Text files %_date >> filelist
dir *.txt >> filelist
```

Using command grouping, these commands can be written much more simply. Enter this example on one line:

```
(echo Data files %_date & dir *.dat & echo. & echo Text files %_date &
   dir *.txt) > filelist
```

The redirection, which appears outside the parentheses, applies to all the commands within the parentheses. Because the redirection is performed only once, the commands will run slightly faster than if each command was entered separately. The same approach can be used for input redirection and for piping (see Redirection and Piping).

You can also use command grouping in a batch file or at the prompt to split commands over several lines. This last example is like the redirection example above, but is entered at the prompt. Note the "More?" prompt after each incomplete line.   None of the commands are executed until the command group is completed with the closing parenthesis.   This example does **not** have to be entered on one line:

```
[c:\] (echo Data files %_date
More? dir *.dat
More? echo.
More? echo Text files %_date
More? dir *.txt) > filelist
[c:\]
```

A group of commands in parentheses is like a long command line.   The total length of the group may not exceed 2,047 characters, whether the commands are entered from the prompt, an alias, or a batch file. The limit **includes** the space required to expand aliases and environment variables used within the group. In addition, each line you type at the normal prompt or the **More?** prompt, and each individual command within the line, must meet the usual length limit of 1,023 characters.

Escape Character

4NT recognizes a user-definable escape character. This character gives the following character a special meaning; it is **not** the same as the ASCII ESC that is often used in ANSI and printer control sequences.

The default escape character is a caret [**^**].

If you don't like using the default escape character, you can pick another character using the SETDOS /E command, the Options 1 page of the OPTION dialogs, or the EscapeChar directive in your *.INI* file. If you plan to share aliases or batch files between 4NT and 4DOS, 4OS2, or Take Command, see Special Character Compatibility for details about choosing compatible escape characters for two or more products.

Ten special characters are recognized when they are preceded by the escape character. The combination of the escape character and one of these characters is translated to a single character, as shown below. These are primarily useful for redirecting codes to the printer; **^e** is also useful to generate ANSI "escape sequences" in your PROMPT, ECHO, or other output commands. The special characters which can follow the escape character are:

b	backspace
c	comma
e	the ASCII ESC character (ASCII 27)
f	form feed
k	back quote
n	line feed
q	double quote
r	carriage return
s	space
t	tab character

If you follow the escape character with any other character, the escape character is removed and the second character is copied directly to the command line. This allows you to suppress the normal meaning of special characters (such as **? * / \ | " ` > <** and **&**). For example, to display a message containing a **>** symbol, which normally indicates redirection:

```
[c:\] echo 2 is ^> 4
```

To send a form feed followed by the sequence ESC Y to the printer, you can use this command:

```
[c:\] echos ^f^eY > prn
```

The escape character has an additional use when it is the last character on any line of a *.BAT or .BTM* batch file. 4NT recognizes this use of the escape character to signal line continuation: the command processor removes the escape character and appends the next line to the current line before executing it.

Aliases

Much of the power of 4NT comes together in **aliases**, which give you the ability to create your own commands. An alias is a name that you select for a command or group of commands. Simple aliases substitute a new name for an existing command. More complex aliases can redefine the default settings of internal or external commands, operate as very fast in-memory batch files, and perform commands based on the results of other commands.

This section will show you some examples of the power of aliases. See the ALIAS command for complete details about writing your own aliases.

The simplest type of alias gives a new name to an existing command. For example, you could create a command called ROOT which uses CD to switch to the root directory this way:

```
[c:\] alias root = cd \
```

After the alias has been defined this way, every time you type the command ROOT, you will actually execute the command CD \.

Aliases can also create customized versions of commands. For example, the DIR command can sort a directory in various ways. You can create an alias called DE that means "sort the directory by filename extension, and pause after each page while displaying it" like this:

```
[c:\] alias de = dir /oe /p
```

Aliases can be used to execute sequences of commands as well. The following command creates an alias called W which saves the current drive and directory, changes to the WP directory on drive C, runs the program *E:\WP60\WP.EXE*, and, when the program terminates, returns to the original drive and directory:

```
[c:\] alias w = `pushd c:\wp & e:\wp60\wp.exe & popd`
```

This alias is enclosed in back-quotes because it contains multiple commands. You must use the back-quotes whenever an alias contains multiple commands, environment variables, parameters (see below), redirection, or piping. See the ALIAS command for full details.

Aliases can be nested, that is, one alias can invoke another. For example, the alias above could also be written as:

```
[c:\] alias wp = e:\wp60\wp.exe
[c:\] alias w = `pushd c:\wp & wp & popd`
```

If you enter W as a command, the command processor will execute the PUSHD command, detect that the next command (WP) is another alias, and execute the program *E:\WP60\WP.EXE*, and – when the program exits – return to the first alias, execute the POPD command, and return to the prompt.

You can use aliases to change the default options for both internal commands and external commands. Suppose that you always want the DEL command to prompt before it erases a file:

```
[c:\] alias del = *del /p
```

An asterisk [*] is used in front of the second "del" to show that it is the name of an internal command, not an alias. See the ALIAS command for more information about this use of the asterisk.

You may have a program on your system that has the same name as an internal command. Normally, if you type the command name, you will start the internal command rather than the program you desire, unless you explicitly add its full path on the command line.   For example, if you have a program named *LIST.COM* in the *C:\UTIL* directory, you could run it with the command *C:\UTIL\LIST.COM*.   However, if you simply type LIST, the internal LIST command will be invoked instead.   Aliases give you two ways to get around this problem.

First, you could define an alias that runs the program in question, but with a different name:

```
[c:\] alias l = c:\util\list.com
```

Another approach is to rename the internal command and use the original name for the external program. The following example renames the LIST command as DISPLAY and then uses a second alias to run *LIST.COM* whenever you type LIST:

```
[c:\] alias display = *list
[c:\] alias list = c:\util\list.com
```

You can also assign an alias to a key, so that every time you press the key, the command will be invoked. You do so by naming the alias with an at sign [**@**] followed by a key name.   After you enter this next example, you will see a 2-column directory with paging whenever you press **Shift-F5**, then **Enter**:

```
[c:\] alias @Shift-F5 = *dir /2/p
```

This alias will put the DIR command on the command line when you press **Shift-F5**, then wait for you to enter file names or additional switches.   You must press **Enter** when you are ready to execute the command. To execute the command immediately, without displaying it on the command line or waiting for you to press **Enter**, use two at signs at the start of the alias name:

```
[c:\] alias @@Shift-F5 = *dir /2/p
```

The next example clears the screen whenever you press **Alt-F1**:

```
[c:\] alias @@Alt-F1 = cls
```

Aliases have many other capabilities as well.   This example creates a simple command-line calculator using the @EVAL function.   Once you have entered the example, you can type CALC 4*19, for example, and you will see the answer:

```
[c:\] alias calc = `echo The answer is:  %@eval[%&]`
```

Our last example in this section creates an alias called IN.   It will temporarily change directories, run an internal or external command, and then return to the current directory when the command is finished:

```
[c:\] alias in = `pushd %1 & %2$ & popd`
```

Now if you type:

```
[c:\] in c:\letters wp letter.txt
```

you will change to the *C:\LETTERS* subdirectory, execute the command WP *LETTER.TXT* and then return to the current directory.

The above example uses two parameters: %1 means the first argument on the command line, and %2$ means the second and all subsequent arguments.   Parameters are explained in detail under the ALIAS command.

Your copy of 4NT includes a sample alias file called *ALIASES* which contains several useful aliases and demonstrates many alias techniques.   See the ALIAS and UNALIAS commands for more information and examples.   Also see Using Aliases in Batch Files.

Batch Files

A batch file is a file that contains a list of commands to execute. 4NT reads and interprets each line as if it had been typed at the keyboard. Like aliases, batch files are handy for automating computing tasks. Unlike aliases, batch files can be as long as you wish. Batch files take up separate disk space for each file, and can't usually execute quite as quickly as aliases, since they must be read from the disk.

The topics included in this section are:

.BAT, *.CMD*, and *.BTM* Files

Echoing in Batch Files

Batch File Parameters

Using Environment Variables

Batch File Commands

Interrupting a Batch File

Automatic Batch Files (4START and 4EXIT)

Detecting 4NT

Using Aliases In Batch Files

Debugging Batch Files

Batch File String Processing

Batch File Line Continuation

Batch File Compression

Special Character Compatibility

Command Parsing

Argument Quoting

REXX Support

EXTPROC Support

.BAT, .CMD, and .BTM Files

A batch file can run in two different modes. In the first, traditional mode, each line of the batch file is read and executed individually. In the second mode, the entire batch file is read into memory at once. The second mode can be 5 to 10 times faster, especially if most of the commands in the batch file are internal commands. However, only the first mode can be used for self-modifying batch files (which are rare).

The batch file's extension determines its mode. Files with a *.CMD* extension are run in the slower, traditional mode. Files with a *.BTM* extension are run in the faster, more efficient mode. You can change the execution mode inside a batch file with the LOADBTM command.

Using .BAT Files

In most cases your batch files will be stored as *.CMD* or *.BTM* files. However, you may also choose to use some *.BAT* files, especially if you are moving from DOS to Windows NT. If you do, you you need to be aware of the way 4NT executes *.BAT* files, which is slightly different from the method used by *CMD.EXE*.

CMD.EXE passes all *.BAT* files to Windows NT's **DOS** command processor, typically *COMMAND.COM*, for execution (yes, there is a DOS command processor in Windows NT!). *COMMAND.COM* handles a few DOS-related commands, but passes most internal commands to a second copy of *CMD.EXE* so that they are executed in the Windows NT environment. This convoluted system allows you to load memory-resident DOS programs (TSRs), and run other programs which use them, all from the same *.BAT* file. However, it reduces performance for all *.BAT* files in order to support those rare files which load DOS TSRs under Windows NT.

4NT does not use this system; it executes *.BAT* files in the normal way, just like *.CMD* and *.BTM* files. This works better for most files, but may render DOS TSRs loaded from a *.BAT* file ineffective because other commands in the file are not executed in DOS-based environment.

In most cases this difference will not affect your *.BAT* files, because you will not be loading DOS TSRs in Windows NT. If you do need to load TSRs from *.BAT* files, we recommend that you obtain a copy of our DOS command processor, 4DOS, start it from your Windows NT desktop, and run the *.BAT* files from a 4DOS session (you could also use a *CMD.EXE* session, but of course the *.BAT* files then cannot use 4DOS or 4NT features). While we do not generally recommend using 4DOS under Windows NT, it can work well in this specific situation.

Echoing in Batch Files

By default, each line in a batch file is displayed or "echoed" as it is executed. You can change this behavior, if you want, in several different ways:

Any batch file line that begins with an [**@**] symbol will not be displayed.

The display can be turned off and on within a batch file with the ECHO OFF and ECHO ON commands.

The default setting can be changed with the SETDOS /V command, on the Options 1 page of the OPTION dialogs, or the BatchEcho directive in the *.INI* file.

For example, the following line turns off echoing inside a batch file. The [**@**] symbol keeps the batch file from displaying the ECHO OFF command:

```
@echo off
```

4NT also has a command line echo that is unrelated to the batch file echo setting. See ECHO for details about both settings.

Batch File Parameters

Like aliases and application programs, batch files can examine the command line that is used to invoke them. The command tail (everything on the command line after the batch file name) is separated into individual **parameters** (also called **arguments** or **batch variables**) by scanning for the spaces, tabs, and commas that separate the parameters. A batch file can work with the individual parameters or with the command tail as a whole.

These parameters are numbered from **%1** to **%127**. **%1** refers to the first parameter on the command line, **%2** to the second, and so on. It is up to the batch file to determine the meaning of each parameter. You can use quotation marks to pass spaces, tabs, commas, and other special characters in a batch file parameter; see Argument Quoting for details.

Parameters that are referred to in a batch file, but which are missing on the command line, appear as empty strings inside the batch file. For example, if you start a batch file and put two parameters on the command line, any reference in the batch file to **%3**, or any higher-numbered parameter, will be interpreted as an empty string.

A batch file can also work with three special parameters: **%0** contains the name of the batch file as it was entered on the command line, **%#** contains the number of command line arguments, and **%n$** contains the complete command-line tail starting with argument number "**n**" (for example, **%3$** means the third parameter and all those after it). The default value of "**n**" is 1, so **%$** contains the entire command tail. The values of these special parameters will change if you use the SHIFT command.

By default, 4DOS uses an ampersand [**&**] instead of a dollar sign [**$**] to indicate the remainder of the command tail. For example, **%&** means all the parameters, and **%2&** means the second parameter and all those after it. If you want to share batch files or aliases between multiple products, see Special Character Compatiblility for information on selecting compatible parameter characters for all products.

For example, if your batch file interprets the first argument as a subdirectory name then the following line would move to the specified directory:

```
        cd %1
```

A friendlier batch file would check to make sure the directory exists and take some special action if it doesn't:

```
        iff isdir %1 then & cd %1
        else & echo Subdirectory %1 does not exist! & quit
        endiff
```

(see the IF and IFF commands).

Batch files can also use environment variables, internal variables, and variable functions.

Using Environment Variables

Batch files can also use environment variables, internal variables, and variable functions. You can use these variables and functions to determine system status (*e.g.*, the type of CPU in the system), resource levels (*e.g.*, the amount of free disk space), file information (*e.g.*, the date and time a file was last modified), and other information (*e.g.*, the current date and time). You can also perform arithmetic operations (including date and time arithmetic), manipulate strings and substrings, extract parts of a filename, and read and write files.

To create temporary variables for use inside a batch file, just use the SET command to store the information you want in an environment variable. Pick a variable name that isn't likely to be in use by some other program (for example, PATH would be a bad choice), and use the UNSET command to remove these variables from the environment at the end of your batch file. You can use SETLOCAL and ENDLOCAL to create a "local" environment so that the original environment will be restored when your batch file is finished.

Environment variables used in a batch file may contain either numbers or text. It is up to you to keep track of what's in each variable and use it appropriately; if you don't (for example, if you use %@EVAL to add a number to a text string), you'll get an error message.

Batch File Commands

Several 4NT commands are particularly suited to batch file processing. Here is a list of some of the commands you might find most useful:

BEEP produces a sound of any pitch and duration through the computer's speaker.

CALL executes one batch file from within another.

CANCEL terminates all batch file processing.

CLS and **COLOR** set the screen display colors.

DO starts a loop. The loop can be based on a counter, or on a conditional test like those used in IF and IFF.

DRAWBOX draws a box on the screen.

DRAWHLINE and **DRAWVLINE** draw horizontal and vertical lines on the screen.

ECHO and **ECHOS** print text on the screen (the text can also be redirected to a file or device). **ECHOERR** and **ECHOSERR** print text to the standard error device.

GOSUB executes a subroutine inside a batch file. The **RETURN** command terminates the subroutine.

GOTO branches to a different location in the batch file.

FOR executes commands for each file that matches a set of wildcards, or each entry in a list.

IF and **IFF** execute commands based on a test of string or numeric values, program exit codes, or other conditions.

INKEY and **INPUT** collect keyboard input from the user and store it in environment variables.

LOADBTM changes the batch file operating mode.

ON initializes error handling for Ctrl-C / Ctrl-Break, or for program and command errors.

PAUSE displays a message and waits for the user to press a key.

QUIT ends the current batch file and optionally returns an exit code.

REM places a remark in a batch file.

SCREEN positions the cursor on the screen and optionally prints a message at the new location.

SCRPUT displays a message in color.

SETLOCAL saves the current disk drive, default directory, environment, alias list, and special character settings. **ENDLOCAL** restores the settings that were saved.

SHIFT changes the numbering of the batch file parameters.

START starts another session or window in certain multitasking environments.

SWITCH selects a group of statements to execute based on the value of a variable.

TEXT displays a block of text. **ENDTEXT** ends the block.

TIMER starts or reads a stopwatch.

VSCRPUT displays a vertical message in color.

These commands, along with the internal variables and variable functions, make the enhanced batch file language extremely powerful. Your copy of 4NT includes a sample batch file, in the file *EXAMPLES.BTM,* that demonstrates some of the things you can do with batch files.

Interrupting a Batch File

You can usually interrupt a batch file by pressing **Ctrl-C** or **Ctrl-Break**. Whether and when these keystrokes are recognized will depend on whether the command processor or an application program is running, how the application (if any) was written, and whether the ON BREAK command is in use.

If 4NT detects a Ctrl-C or Ctrl-Break (and ON BREAK is not in use), it will display a prompt, for example:

```
Cancel batch job C:\CHARGE.BTM ? (Y/N/A) :
```

Enter **N** to continue, **Y** to terminate the current batch file and continue with any batch file which called it, or **A** to end all batch file processing regardless of the batch file nesting level. Answering **Y** is similar to the QUIT command; answering **A** is similar to the CANCEL command.

Automatic Batch Files

4NT supports "automatic" batch files, files that run without your intervention, as long as the command processor can find them.

Each time 4NT starts as either a primary or a secondary shell, it looks for an automatic batch file called *4START.BTM* or *4START.CMD*. If the *4START* batch file is not in the same directory as 4NT itself, you should use the Startup page of the OPTION dialogs or the 4StartPath directive in your *.INI* file to specify its location. *4START* is optional, so 4NT will not display an error message if it cannot find the file.

4START is a convenient place to change the color or content of the prompt for each shell, LOG the start of a shell, or put other special startup or configuration commands. *4START* is also a good place to set aliases and environment variables.

The entire startup command line passed to the command processor is available to *4START* via batch file parameters (%1, %2, etc.). This can be useful if you want to see the command line passed to a secondary shell by an application. For example, to pause if any parameters are passed to a secondary shell you could include this command in *4START* (enter this on one line):

```
if "%1" != "" .and. "%_shell" gt 0 pause Starting shell %_shell with
parameters [%$]
```

Whenever a 4NT shell ends, it runs an automatic batch file called *4EXIT.BTM* or *4EXIT.CMD*. This file, if you use it, should be in the same directory as your *4START* batch file. Like *4START*, *4EXIT* is optional. It is not necessary in most circumstances, but it is a convenient place to put commands to save information such as a history list before a shell ends, or LOG the end of the shell.

Pipes, Transient Sessions, and 4START

When you set up the 4START file, remember that it is executed **every** time 4NT starts, including when running a pipe, or a transient copy of the command processor started with the ***/C*** startup option (see Starting 4NT for details on /C).

For example, suppose you enter a command line like this, which uses a pipe:

```
c:\data> myprog | sort > out.txt
```

Normally this command would create the output file *C:\DATA\OUT.TXT*. However, if you have a 4START file which changes to a different directory, the output file will be written there — not in *C:\DATA*.

This is because the command processor starts a second copy of itself to run the commands on the right hand side of the pipe, and that new copy runs 4START before processing the commands from the pipe. If 4START changes directories, the command from the pipe will be executed in the new directory.

The same problem can occur if you use a transient session started with /C to run an individual command, then exit — the session will execute in the directory set by 4START, not the directory in which it was originally started. For example, suppose you set up a Windows desktop object with a command line like this, which starts a transient session:

```
Command:                d:\4nt\4nt.exe /c list myfile.txt
Working Directory:c:\data
```

Normally this command would LIST the file *C:\DATA\MYFILE.TXT*. However, if 4START changes to a different directory, The command processor will look for *MYFILE.TXT* there — not in *C:\DATA*.

Similarly, any changes to environment variables or other settings in 4START will affect all copies of the command processor, including those used for pipes and transient sessions.

You can work around these potential problems with the IF or IFF command and the internal variables _PIPE and _TRANSIENT. For example, to skip all 4START processing when running in a pipe or transient session, you could use a command like this at the beginning of 4START:

```
if %_pipe != 0 .or. %_transient != 0 quit
```

Detecting 4NT

From a batch file, you can determine if 4NT (or another JP Software command processor) is loaded by testing for the variable function @EVAL, with a test like this:

```
      if "%@eval[2+2]" == "4" echo 4NT is loaded!
```

This test can never succeed in *CMD.EXE.* Other variable functions could be used for the same purpose.

Using Aliases in Batch Files

One way to simplify batch file programming is to use aliases to hide unnecessary detail inside a batch file. For example, suppose you want a batch file to check for certain errors, and display a message and exit if one is encountered. This example shows one way to do so:

```
setlocal
unalias *
alias error `echo. & echo ERROR: %$ & goto dispmenu`
alias fatalerror `echo. & echo FATAL ERROR: %$ & quit`
alias in `pushd %1 & %2$ & popd`
if not exist setup.btm fatalerror Missing setup file!
call setup.btm
cls
:dispmenu
text
          1. Word Processing
          2. Spreadsheet
          3. Communications
          4. Exit
endtext
echo.
inkey Enter your choice:  %%userchoice
switch %userchoice
case 1
   input Enter the file name:  %%fname
   if not exist fname error File does not exist
   in d:\letters c:\wp60\wp.exe
case 2
   in d:\finance c:\quattro\q.exe
case 3
   in d:\comm c:\comsw\pcplus.exe
case 4
   goto done
default
  error Invalid choice, try again
endswitch
goto dispmenu
:done
endlocal
```

The first alias, ERROR, simply displays an error message and jumps to the label DISPMENU to redisplay the menu. The "%$" in the second ECHO command displays all the text passed to ERROR as the content of the message. The similar FATALERROR alias displays the message, then exits the batch file.

The last alias, IN, expects 2 or more command-line arguments. It uses the first as a new working directory and changes to that directory with a PUSHD command. The rest of the command line is interpreted as another command plus possible command line parameters, which the alias executes. This alias is used here to switch to a directory, run an application, and switch back. It could also be used from the command line.

The following lines print a menu on the screen and then get a keystroke from the user and store the keystroke in an environment variable called *userchoice*. Then the SWITCH command is used to test the user's keystroke and decide what action to take.

There's another side to aliases in batch files.   If you're going to distribute your batch files to others, you need to remember that they may have aliases defined for the commands you're going to use.   For example if the user has aliased CD to CDD and you aren't expecting this, your file may not work as you intended.   There are two ways to address this problem.

First, you can use SETLOCAL, ENDLOCAL, and UNALIAS to clear out aliases before your batch file starts and restore them at theend, as we did in the previous example.   Remember that SETLOCAL and ENDLOCAL will save and restore not only the aliases but also the environment, the current drive and directory, and various special characters.

If this method isn't appropriate or necessary for the batch file you're working on, you can also use an asterisk [*****] before the name of any command.   The asterisk means the command that follows it should not be interpreted as an alias.   For example the following command redirects a list of file names to the file *FILELIST*:

```
dir /b > filelist
```

However, if the user has redefined DIR with an alias this command may not do what you want.   To get around this just use:

```
*dir /b > filelist
```

The same can be done for any command in your batch file.   If you use the asterisk, it will disable alias processing, and the rest of the command will be processed normally as an internal command, external command, or batch file.   Using an asterisk before a command will work whether or not there is actually an alias defined with the same name as the command.   If there is no alias with that name, the asterisk will be ignored and the command will be processed as if the asterisk wasn't there.

Debugging Batch Files

4NT includes a built-in batch file debugger, invoked with the SETDOS /Y1 command. The debugger allows you to "single-step" through a batch file line by line, with the file displayed in a popup window as it executes. You can execute or skip the current line, continue execution with the debugger turned off, view the fully-expanded version of the command line, or exit the batch file. The batch debugger can also pop up a separate window to view current environment variables or aliases so you can check their values during execution, and can pop up the LIST command to display the contents of any file.

To start the debugger, insert a SETDOS /Y1 command at the beginning of the portion of the batch file you want to debug, and a SETDOS /Y0 command at the end. You can also invoke SETDOS /Y1 from the prompt, but because the debugger is automatically turned off whenever the command processor returns to the prompt, you must enter the SETDOS command and the batch file name on the same line, for example:

```
[c:\] setdos /y1 & mybatch.btm
```

If you use the debugger regularly you may want to define a simple alias to invoke it, for example:

```
[c:\] alias trace `setdos /y1 & %$`
```

This alias simply enables the debugger, then runs whatever command is passed to it. You can use the alias to debug a batch file with a command like this:

```
[c:\] trace mybatch.btm
```

When the debugger is running you can control its behavior with keystrokes. Debugging continues after each keystroke unless otherwise noted:

T(race), **Enter**, or **F8**	Execute the current command. If it calls a subroutine with GOSUB, or another batch file with CALL, single-step into the called subroutine or batch file.
S(tep) or **F10**	Execute the current command, but execute any subroutine or CALLed batch file without single-stepping.
J(ump)	Skip the current command and proceed to the next command.
X (Expand)	Display the next command to be executed, after expansion of aliases and environment variables.
L(ist)	Prompt for a file name and then view the file with the LIST command.
V(ariables)	Open a popup window to display the current environment, in alphabetical order.
A(liases)	Open a popup window to display the current aliases, in alphabetical order.
O(ff) or **Esc**	Turn off the debugger and continue with the remainder of the batch file.
Q(uit)	Quit the debugger and the current batch file, without executing the remainder of the file.

The debugger highlights each line of the batch file as it is executed. It executes the commands on the line one at a time, so when a line contains more than one command, the highlight will not move as each command is executed. To see the individual commands, use the **X** key to expand each command before it is executed.

If you use a "prefix" command like EXCEPT, FOR, GLOBAL, or SELECT, the prefix command is considered one command, and each command it invokes is another. For example, this command line executes four commands — the FOR and three ECHO commands:

```
      for %x in (a b c) do echo %x
```

You cannot use the batch debugger with REXX files or EXTPROC files. It can only be used with normal 4NT batch files.

The debugger gives you a detailed, step-by-step view of batch file execution, and will help solve particularly difficult batch file problems. However, in some cases you will find it easier to diagnose these problems with techniques that allow you to review what is happening at specific points in the batch file without stepping through each line individually.

There are several tricks you can use for this purpose. Probably the simplest is to turn ECHO on at the beginning of the file while you're testing it, or use SETDOS /V2 to force ECHO on even if an ECHO OFF command is used in the batch file. This will give you a picture of what is happening as the file is executed, without stopping at each line. It will make your output look messy of course, so just turn it off once things are working. You can also turn ECHO on at the beginning of a group of commands you want to "watch", and off at the end, just by adding ECHO commands at the appropriate spots in your file.

If an error occurs in a batch file, the error message will display the name of the file, the number of the line that contained the error, and the error itself. For example:

```
      e:\test.bat [3] Invalid parameter "/d"
```

tells you that the file *E:\TEST.BAT* contains an error on line 3. The first line of the batch file is numbered 1.

Another trick, especially useful in a fast-moving batch file or one where the screen is cleared before you can read messages, is to insert PAUSE commands wherever you need them in order to be able to watch what's happening. You can also use an ON ERRORMSG command to pause if an error occurs, then continue with the rest of the file (the first command below), or to quit if an error occurs (the second command):

```
      on errormsg pause
      on errormsg quit
```

If you can't figure out how your aliases and variables are expanded, try turning LOG on at the start of the batch file. LOG keeps track of all commands after alias and variable expansion are completed, and gives you a record in a file that you can examine after the batch file is done. You must use a standard LOG command; LOG /H (the history log) does not work in batch files.

You may also want to consider using redirection to capture your batch file output. Simply type the batch file name followed by the redirection symbols, for example:

```
      [c:\] mybatch >& testout
```

This records all batch file output, including error messages, in the file *TESTOUT*, so you can go back and examine it. If you have ECHO ON in the batch file you'll get the batch commands intermingled with the output, which can provide a very useful trace of what's happening. Of course, output from full-screen commands and programs that don't write to the standard output devices can't be recorded, but you can still gain a lot of useful information if your batch file produces any output.

If you're using redirection to see the output, remember that any prompts for input will probably go to the output file and not to the screen, so you need to know in advance the sequence of keystrokes required to get through the entire batch file, and enter them by hand.   You can also use the TEE command to both view the output while the batch file is running and save it in a file for later examination.

String Processing

As you gain experience with batch files, you're likely to find that you need to manipulate text strings. You may need to prompt a user for a name or password, process a list of files, or find a name in a phone list. All of these are examples of string processing – the manipulation of lines of readable text.

4NT include several features that make string processing easier. For example, you can use the INKEY and INPUT commands for user input; the ECHO, SCREEN, SCRPUT, and VSCRPUT commands for output; and the FOR command or the @FILEREAD function to scan through the lines of a file. In addition, variable functions offer a wide range of string handling capabilities.

For example, suppose you need a batch file that will prompt a user for a name, break the name into a first name and a last name, and then run a hypothetical LOGIN program. LOGIN expects the syntax **/F:first /L:last** with both the first and last names in upper case and neither name longer than 8 characters. Here is one way to write such a program:

```
@echo off
setlocal
unalias *
input Enter your name (no initials):  %%name

set first=%@word[0,%name]
set flen=%@len[%first]
set last=%@word[1,%name]
set llen=%@len[%last]

iff %flen gt 8 .or. %llen gt 8 then
      echo First or last name too long
      quit
endiff

login /F:%@upper[%first] /L:%@upper[%last]
endlocal
```

The SETLOCAL command at the beginning of this batch file saves the environment and aliases. Then the UNALIAS * command removes any existing aliases so they won't interfere with the behavior of the commands in the remainder of the batch file. The first block of lines ends with an INPUT command which asks the user to enter a name. The user's input is stored in the environment variable NAME.

The second block of lines extracts the user's first and last names from the NAME variable and calculates the length of each. It stores the first and last name, along with the length of each, in additional environment variables. Note that the @WORD function numbers the first word as 0, not as 1.

The IFF command in the third block of lines tests the length of both the first and last names. If either is longer than 8 characters, the batch file displays an error message and ends. Finally, in the last block, the batch file executes the LOGIN program with the appropriate parameters, then uses the ENDLOCAL command to restore the original environment and alias list. At the same time, ENDLOCAL discards the temporary variables that the batch file used (NAME, FIRST, FLEN, etc.).

When you're processing strings, you also need to avoid some common traps. The biggest one is handling special characters. Suppose you have a batch file with these two commands, which simply accept a string and display it:

```
input Enter a string:  %%str
echo %str
```

Those lines look safe, but what happens if the user enters the string "some > none" (without the quotes). After the string is placed in the variable STR, the second line becomes:

```
echo some > none
```

The ">" is a redirection symbol, so the line echoes the string "some" and redirects it to a file called NONE – probably not what you expected. You could try using quotation marks to avoid this kind of problem (see Argument Quoting), but that won't quite work. If you use back-quotes (ECHO `%STR`), the command will echo the four-character string %STR. Environment variable names are not expanded (replaced by their contents) when they are inside back-quotes.

If you use double quotes (ECHO "%STR"), the string entered by the user will be displayed properly, and so will the quotation marks. With double quotes, the output would look like this:

```
"some > none"
```

As you can imagine, this kind of problem becomes much more difficult if you try to process text from a file. Special characters in the text can cause all kinds of confusion in your batch files. Text containing back-quotes, double quotes, or redirection symbols can be virtually impossible to handle correctly.

One way to overcome these potential problems is to use the SETDOS /X command to temporarily disable redirection symbols and other special characters. The two-line batch file above would be a lot more likely to produce the expected results if it were rewritten this way:

```
setdos /x-15678
input Enter a string:  %%str
echo %str
setdos /x0
```

The first line turns off alias processing and disables several special symbols, including the command separator (see Multiple Commands) and all redirection symbols. Once the string has been processed, the last line re-enables the features that were turned off in the first line.

If you need advanced string processing capabilities beyond those provided by 4NT, you may want to consider using the REXX language. Our products support external REXX programs for this purpose.

Batch File Line Continuation

4NT will combine multiple lines in the batch file into a single line for processing when you include the escape character as the very last character of each line to be combined (except the last). The default escape character is a caret [**^**]. For example:

```
[c:\] echo The quick brown fox jumped over the lazy^
sleeping^
dog. > alphabet
```

You cannot use this technique to extend a batch file line beyond the normal line length limit of 1,023 characters.

Batch File Compression

You can compress your batch files with a program called *BATCOMP.EXE*, which is distributed with 4NT. This program condenses batch files by about a third and makes them unreadable with the LIST command and similar utilities. Compressed batch files run at approximately the same speed as regular *.BTM* files.

You may want to consider compressing batch files if you need to distribute them to others and keep your original code secret or prevent your users from altering them. You may also want to consider compressing batch files to save some disk space on the systems where the compressed files are used. (However, you will not save space if you keep your compressed batch files on a disk compressed with a program like DBLSPACE, Stacker, or SuperStor.)

The full syntax for the batch compression program is

```
        BATCOMP [/O] input file [output file]
```

You must specify the full name of the input file, including its extension, on the BATCOMP command line. If you do not specify the output file, BATCOMP will use the same base name as the input file and add a *.BTM* extension. BATCOMP will also add a *.BTM* extension if you specify a base name for the output file without an extension. For example, to compress *MYBATCH.CMD* and save the result as *MYBATCH.BTM*, you can use any of these three commands:

```
        [c:\] batcomp mybatch.cmd
        [c:\] batcomp mybatch.cmd mybatch
        [c:\] batcomp mybatch.cmd mybatch.btm
```

If the output file (*MYBATCH.BTM* in the examples above) already exists, BATCOMP will prompt you before overwriting the file. You can disable the prompt by including **/O** on the BATCOMP command line immediately before the input file name. Even if you use the /O option, BATCOMP will not compress a file into itself.

JP Software does not provide a decompression utility to uncompress batch files. If you use *BATCOMP.EXE*, make sure that you also keep a copy of the original batch file for future inspection or modification.

You can adopt one of two strategies for keeping track of your original source files and compressed batch files. First, you may want to create the source files with a traditional *.BAT* or *.CMD* extension and reserve the *.BTM* extension for compressed batch files. The advantage of this approach is that you can modify and test the uncompressed versions at any time, although they will run in the slower, traditional mode unless they begin with a LOADBTM command.

If you prefer, you can use a *.BTM* extension for both the source and compressed files. In this case you will have to use a different directory or a different base name for each file. For example, you might use *SOURCE\MYBATCH.BTM* for the source file and *COMP\MYBATCH.BTM* for the compressed version, or use *MYBATCHS.BTM* for the source file and *MYBATCH.BTM* for the compressed file (however, the latter approach may make it more difficult to keep track of the correspondence between the source file and the compressed file).

BATCOMP is a DOS and OS/2 character-mode application designed to run in any environment where our command processors run. Each of our command processors includes the same version of *BATCOMP.EXE*, and a batch file compressed with any copy of BATCOMP can be used with any current JP Software command processor.

If you plan to distribute batch files to users of different platforms, see Special Character Compatibility.

Special Character Compatibility

If you use two or more of our products, or if you want to share aliases and batch files with users of different products, you need to be aware of the differences in three important characters: the Command Separator (see Multiple Commands), the Escape Character (see Escape Character), and the Parameter Character (see Batch File Parameters).

The default values of each of these characters in each product is shown in the following chart:

Product	Separator	Escape	Parameter
4DOS, Take Command/16	**^**		**&**
4NT, 4OS2, Take Command/32, Take Command for OS/2	**&**	**^**	**$**

(The up-arrow [] represents the ASCII Ctrl-X character, numeric value 24.)

In your batch files and aliases, and even at the command line, you can smooth over these differences in three ways:

» Select a consistent set of characters from the Options 1 page of the OPTION dialogs, or with *.INI* file configuration directives. For example, to set the 4NT characters to match 4DOS, use these lines in *4NT.INI*:

```
CommandSep = ^
EscapeChar = 
ParameterChar = &
```

» Use internal variables that contain the current special character, rather than using the character itself (see + and =). For example, this command:

```
if "%1" == "" (echo Argument missing! ^ quit)
```

will only work if the command separator is a caret. However, this version works regardless of the current command separator:

```
if "%1" == "" (echo Argument missing! %+ quit)
```

» In a batch file, use the SETLOCAL command to save the command separator, escape character, and parameter character when the batch file starts. Then use SETDOS as described below to select the characters you want to use within the batch file. Use an ENDLOCAL command at the end of the batch file to restore the previous settings.

You can also use the SETDOS command to change special characters on the command line. However, when setting new special character values on the command line you must take into account the possibility that one of your new values will have a current meaning that causes problems with the setting. For example, this command:

```
[c:\] setdos /p&
```

would **not** set the parameter character to an ampersand [**&**] in 4NT if the standard 4NT special characters were currently in effect. The **&** would be seen as a command separator, and would terminate the SETDOS command before the parameter character was set. To work around this, use the escape character variable **%=** before each setting to ensure that the following character is not treated with any special meaning.

For example, the following sequence of commands in a batch file will always set the special characters correctly to their standard 4DOS values, no matter what their current setting, and will restore them when the batch file is done:

```
setlocal
setdos /c%=^ /e%= /p%=&
.....
endlocal
```

A similar sequence can be used to select the standard 4OS2 and 4NT characters, regardless of the current settings:

```
setlocal
setdos /c%=& /e%=^ /p%=$
.....
endlocal
```

Command Parsing

Whenever you type something at the command line and press the **Enter** key, or include a command in a batch file, you have given a command to 4NT, which must figure out how to execute your command. If you understand the general process that is used, you will be able to make the best use of the commands. Understanding these steps can be especially helpful when working with complex aliases or batch file commands.

To decide what activity to perform, the command processor goes through several steps. Before it starts, it writes the entire command line (which may contain multiple commands) to the history log file if history logging has been enabled with the LOG /H command, and the command did not come from a batch file. Then, if the line contains multiple commands, the first command is isolated for processing.

4NT begins by dividing the command into a **command name** and a **command tail**. The command name is the first word in the command; the tail is everything that follows the command name. For example, in the command line

```
dir *.txt /2/p/v
```

the command name is "dir", and the command tail is " *.txt /2/p/v".

Next 4NT tries to match the command name against its list of aliases. If it finds a match between the command name and one of the aliases you've defined, it replaces the command name with the contents of the alias. This substitution is done internally and is not normally visible to you; however, you can view a command line with aliases expanded by pressing **Ctrl-F** after entering the command at the prompt.

If the alias included parameters (%1, %2, etc.), the parameter values are filled in from the text on the command line, and any parameters used in this process are removed from the command line. The process of replacing a command name that refers to an alias with the contents of the alias, and filling in the alias parameters, is called **alias expansion**.

This expansion of an alias creates a new command name: the first word of the alias. This new command name is again tested against the list of aliases, and if a match is found the contents of the new alias is expanded just like the first alias. This process, called **nested alias expansion**, continues until the command name no longer refers to an alias.

If the command name is not an alias or an Internet URL beginning with **http:**, 4NT tries to match the command name with its list of internal commands. If it is unsuccessful, the command processor knows that it will have to search for a batch file or external program to execute your command.

The next step is to locate any batch file or alias parameters, environment variables, internal variables, or variable functions in the command, and replace each one with its value. This process is called **variable expansion**.

The variable expansion process is modified for certain internal commands, like EXCEPT, IF, and GLOBAL. These commands are always followed by another command, so variable expansion takes place separately for the original command and the command that follows it.

Once all of the aliases and environment variables have been expanded, 4NT will echo the complete command to the screen (if command-line echo has been enabled) and write it to the log file (if command logging has been turned on).

Before it can actually execute your command, the command processor must scan the command tail to see if it includes redirection or piping. If so, the proper internal switches are set to send output to an alternate device or to a file, instead of to the screen. A second process is started at this point, if necessary, to receive any piped output.

Finally, it is time to execute the command. If the command name matches an internal command, 4NT will perform the activities you have requested. Otherwise, the command processor searches for an executable (*.COM* or *.EXE*) file, a batch file, or a file with an executable extension that matches the command name.

Once the internal command or external program has terminated, the command processor saves the result or exit code that the command generated, cleans up any redirection that you specified, and then returns to the original command line to retrieve the next command. When all of the commands in a command line are finished, the next line is read from the current batch file, or if no batch file is active, the prompt is displayed.

You can disable and re-enable several parts of command parsing (for example alias expansion, variable expansion, and redirection) with the SETDOS /X command.

Argument Quoting

As it parses the command line, 4NT looks for the ampersand [**&**] command separator, conditional commands (**||** or **&&**), white space (spaces, tabs, and commas), percent signs [**%**] which indicate variables to be expanded, and redirection and piping characters (**>**, **<**, or **|**).

Normally, these special characters cannot be passed to a command as part of an argument. However, you can include any of the special characters in an argument by enclosing the entire argument in single back quotes [**`**] or double quotes [**"**]. Although both back quotes and double quotes will let you build arguments that include special characters, they do not work the same way.

No alias or variable expansion is performed on an argument enclosed in back quotes. Redirection symbols inside the back quotes are ignored. The back quotes are removed from the command line before the command is executed.

No alias expansion is performed on expressions enclosed in double quotes. Redirection symbols inside double quotes are ignored. However, variable expansion **is** performed on expressions inside double quotes. The double quotes themselves will be passed to the command as part of the argument.

For example, suppose you have a batch file *CHKNAME.BTM* which expects a name as its first parameter (%1). Normally the name is a single word. If you need to pass a two-word name with a space in it to this batch file you could use the command:

```
[c:\] chkname `MY NAME`
```

Inside the batch file, %1 will have the value MY NAME, including the space. The back quotes caused 4NT to pass the string to the batch file as a single argument. The quotes keep characters together and reduce the number of arguments in the line.

For a more complex example, suppose the batch file *QUOTES.BAT* contains the following commands:

```
@echo off
echo Arg1 = %1
echo Arg2 = %2
echo Arg3 = %3
```

and that the environment variable FORVAR has been defined with this command:

```
[c:\] set FORVAR=for
```

Now, if you enter the command

```
[c:\] quotes `Now is the time %%forvar` all good
```

the output from *QUOTES.BAT* will look like this:

```
Arg1 = Now is the time %forvar
Arg2 = all
Arg3 = good
```

But if you enter the command

```
[c:\] quotes "Now is the time %%forvar" all good
```

the output from *QUOTES.BAT* will look like this:

```
      Arg1 = "Now is the time for"
      Arg2 = all
      Arg3 = good
```

Notice that in both cases, the quotes keep characters together and reduce the number of arguments in the line.

The following example has 7 command-line arguments, while the examples above only have 3:

```
      [c:\] quotes Now is the time %%forvar all good
```

(The double percent signs are needed in each case because the argument is parsed twice, once when passed to the batch file and again in the ECHO command.)

When an alias is defined in a batch file or from the command line, its argument can be enclosed in back-quotes to prevent the expansion of replaceable parameters, variables, and multiple commands until the alias is invoked.   See ALIAS for details.

You can disable and re-enable back quotes and double quotes with the SETDOS /X command.

REXX Support

REXX is a a powerful file and text processing language developed by IBM, and available on many PC and other platforms. REXX is an ideal extension to the 4NT batch language, especially if you need advanced string processing capabilities.

The REXX language is not built into 4NT, and requires a separate REXX processor. You can purchase add-on REXX software such as Enterprise Alternatives' Enterprise REXX, available for Windows 3.x, Windows 95, and Windows NT; or Quercus's Personal REXX, available for DOS, OS/2, Windows 3.x, Windows 95, and Windows NT. (If you want to learn about or purchase one of these REXX packages, contact JP Software's sales department for more information.)

REXX programs are stored in *.CMD* or *.REX* files. When 4NT loads, it asks Windows NT to locate the Enterprise REXX or Personal REXX libraries. If they are available, 4NT checks each *.CMD* or *.REX* file you execute to see if the first two characters on the first line are [**/***]. If so, it passes the file to Enterprise REXX or Personal REXX for Windows NT for processing.

Enterprise REXX and Personal REXX extend the interface between REXX and the command processor by allowing you to invoke 4NT commands from within a REXX program.

When you send a command from a REXX program back to the command processor to be executed (for example, if you execute a DIR command within a REXX script), the REXX software must use the correct "address" for the command processor. In most cases it is best to use the default address of **CMD**, which is set up automatically by 4NT. If you choose to use an explicit address via the REXX ADDRESS command, you must use **CMD**.

For details on communication between REXX and the command processor, or for more information on any aspect of REXX, see your REXX documentation.

EXTPROC Support

For compatibility with *CMD.EXE*, 4NT offers an external processor (EXTPROC) option for batch files that lets you define an external program to process a particular *.CMD* file.   To identify a *.CMD* file to be used with an external processor, place the string "EXTPROC" as the first word on the first line of the file, followed by the name of the external program that should be called.   4NT will start the program and pass it the name of the *.CMD* file and any command-line arguments that were entered.

For example, suppose *GETDATA.CMD* contains the following lines:

```
EXTPROC D:\DATAACQ\DATALOAD.EXE
OPEN PORT1
READ 4000
DISKWRITE D:\DATAACQ\PORT1\RAW
```

Then if you entered the command:

```
[d:\dataacq] getdata /p17
```

4NT would read the *GETDATA.CMD* file, determine that it began with an EXTPROC command, read the name of the processor program, and then execute the command:

```
D:\DATAACQ\DATALOAD.EXE D:\DATAACQ\GETDATA.CMD /p17
```

The hypothetical *DATALOAD.EXE* program would then be responsible for reopening the *GETDATA.CMD* file, ignoring the EXTPROC line at the start, and interpreting the other instructions in the file. It would also have to respond appropriately to the command-line parameter entered (/p17).

Do not try to use 4NT as the external processor named on the EXTPROC line in the *.CMD* file.   It will interpret the EXTPROC line as a command to re-open themselves.   The result will be an infinite loop that will continue until the computer runs out of resources and locks up.

Environment Variables and Functions

The **environment** is a collection of information about your computer that every program receives. Each entry in the environment consists of a variable name, followed by an equal sign and a string of text. You can automatically substitute the text for the variable name in any command. To create the substitution, include a percent sign [**%**] and a variable name on the command line or in an alias or batch file.

The following environment variables have special meanings in 4NT:

CDPATH

CMDLINE

COLORDIR

COMSPEC

FILECOMPLETION

PATH

PROMPT

4NT also supports two special types of variables. Internal variables are similar to environment variables, but are stored internally within 4NT, and are not visible in the environment. They provide information about your system for use in batch files and aliases. Variable functions are referenced like environment variables, but perform additional functions like file handling, string manipulation and arithmetic calculations.

The SET command is used to create environment variables. For example, you can create a variable named BACKUP like this:

```
[c:\] set BACKUP=*.bak;*.bk!;*.bk
```

If you then type

```
[c:\] del %BACKUP
```

it is equivalent to the following command:

```
del *.bak;*.bk!;*.bk
```

Environment variable names may contain any alphabetic or numeric characters, the underscore character [_], and the dollar sign [**$**]. You can force acceptance of other characters by including the full variable name in square brackets, like this: **%[AB##2]**. You can also "nest" environment variables using square brackets. For example **%[%var1]** means "the contents of the variable whose name is stored in VAR1". A variable referenced with this technique cannot contain more than 255 characters of information. Nested variable expansion can be disabled with the SETDOS /X command.

In addition, 4NT uses the environment to keep track of the default directory on each drive. Other operating systems keep track of the default directory for each drive letter internally; Windows NT does not. 4NT overcomes this incompatibility by saving the default directory for each drive in the environment, using variable names that cannot be accessed by the user. Each variable begins with an equal sign followed by the drive letter and a colon (for example, **=C:**). You cannot view or change these variables with the SET command; they are only available for internal use by 4NT.

In 4NT the size of the environment is set automatically, and increased as needed when you add variables.

Environment variables may contain alias names. The command processor will substitute the variable value for the name, then check for any alias name which may have been included within the variable's value. For example, the following commands would generate a 2-column directory of the *.TXT* files:

```
[c:\] alias d2 dir /2
[c:\] set cmd=d2
[c:\] %cmd *.txt
```

The trailing percent sign that was traditionally required for environment variable names is not usually required in 4NT, which accept any character that cannot be part of a variable name (including a space) as the terminator. However, the trailing percent can be used to maintain compatibility.

The trailing percent sign **is** needed if you want to join two variable values. The following examples show the possible interactions between variables and literal strings. First, create two environment variables called ONE and TWO this way:

```
[c:\] set ONE=abcd
[c:\] set TWO=efgh
```

Now the following combinations produce the output text shown:

```
%ONE%TWO      abcdTWO   ("%ONE%" + "TWO")
%ONE%TWO%     abcdTWO   ("%ONE%" + "TWO%")
%ONE%%TWO     abcdefgh  ("%ONE%" + "%TWO")
%ONE%%TWO%    abcdefgh  ("%ONE%" + "%TWO%")
%ONE%[TWO]    abcd[TWO] ("%ONE%" + "[TWO]")
%ONE%[TWO]%   abcd[TWO] ("%ONE%" + "[TWO]%")
%[ONE]%TWO    abcdefgh  ("%[ONE]" + "%TWO")
%[ONE]%TWO%   abcdefgh  ("%[ONE]" + "%TWO%")
```

If you want to pass a percent sign to a command, or a string which includes a percent sign, you must use two percent signs in a row. Otherwise, the single percent sign will be seen as the beginning of a variable name and will not be passed on to the command. For example, to display the string "We're with you 100%" you would use the command:

```
echo We're with you 100%%
```

You can also use back quotes around the text, rather than a double percent sign. See Argument Quoting for details.

CMDLINE

CMDLINE is the fully expanded text of the currently executing command line.   CMDLINE is set just before invoking any *.COM*, *.EXE*, *.BTM*, *.BAT*, or *.CMD* file.   If a command line is prefaced with an "@" to prevent echoing, it will not be put in CMDLINE, and any previous CMDLINE variable will be removed from the environment.

COLORDIR

COLORDIR controls directory display colors used by DIR and SELECT.   See Color-Coded Directories for a complete description of the format of this variable.

COMSPEC

COMSPEC contains the full path and name of 4NT.   For example, if 4NT is stored in the directory *C:\4NT*, the COMSPEC variable should be set to *C:\4NT\4NT.EXE*.   COMSPEC is used by applications which need to find 4NT to implement a "shell to the command prompt" feature.

You can set the COMSPEC variable by specifying the COMSPEC path on the 4NT startup command line.

FILECOMPLETION

FILECOMPLETION sets the files made available during filename completion for selected commands. See Customizing Filename Completion for a complete description of the format of this variable.

PATH

PATH is a list of directories that 4NT will search for executable files that aren't in the current directory. PATH may also be used by some application programs to find their own files. See the PATH command for a full description of this variable.

PROMPT

PROMPT defines the command-line prompt.   It can be set or changed with the PROMPT command.

Internal Variables

Internal variables are special variables built into 4NT to provide information about your system. They are not actually stored in the environment, but can be used in commands, aliases, and batch files just like any environment variable. The values of these variables are stored internally in 4NT, and cannot be changed with the SET, UNSET, or ESET command. However, you can override any of these variables by defining a new variable with the same name.

The list below gives a one-line description of each variable, and a cross-reference which selects a full screen help topic on that variable. Most of the variables are simple enough that the one-line description is sufficient. However, for those variables marked with an asterisk [*****], the cross-reference topic contains some additional information you may wish to review. You can also obtain help on any variable with a **HELP variable name** command at the prompt (this is why each variable has its own topic, in addition to its appearance in the list below).

See the discussion after the variable list for some additional information, and examples of how these variables can be used. For additional examples see the *EXAMPLES.BTM* file which came with 4NT.

The variables are:

Hardware status

_CPU	CPU type (386, 486, 586)
_KBHIT	Keystroke waiting in buffer (1 or 0)
_NDP	Coprocessor type (0, 387)

Operating system and software status

_ANSI	ANSI status (always 0 in 4NT)
_BOOT	Boot drive letter, without a colon
_CODEPAGE	Current code page number
_COUNTRY	Current country code
_DOS	* Operating system (DOS, OS2, etc.)
_DOSVER	* Operating system version (3.5, etc.)
_MOUSE	Mouse driver flag (always 1 in 4NT)
_WINDIR	Windows NT directory pathname
_WINSYSDIR	Windows NT system directory pathname
_WINTITLE	Current window title
_WINVER	Windows NT version number

Command processor status

_4VER	4NT version (2.5, 2.51, etc.)
_BATCH	Batch nesting level
_BATCHLINE	Batch file line number
_BATCHNAME	Batch file name
_DNAME	Description file name
_HLOGFILE	Current history log file name

_LOGFILE	Current log file name
_PID	4NT process ID (numeric)
_PIPE	Running in a pipe (0 or 1)
_SHELL	Shell level (0, 1, 2, ...)
_TRANSIENT	* Transient shell flag (0 or 1)

Screen, color, and cursor

_BG	Background color at cursor position
_CI	Current text cursor shape in insert mode
_CO	Current text cursor shape in overstrike mode
_COLUMN	Current cursor column
_COLUMNS	Screen width
_FG	Foreground color at cursor position
_ROW	Current cursor row
_ROWS	Screen height

Drives and directories

_CWD	Current drive and directory (d:\path)
_CWDS	Current drive and directory with trailing **** (d:\path\)
_CWP	Current directory (\path)
_CWPS	Current directory with trailing **** (\path\)
_DISK	Current drive (C, D, etc.)
_LASTDISK	Last possible drive (E, F, etc.)

Dates and times

_DATE	* Current date (mm-dd-yy)
_DAY	Day of the month (1 - 31)
_DOW	Day of the week (Mon, Tue, Wed, etc.)
_DOWI	Integer day of the week (1 = Sunday, 2 = Monday, etc.)
_DOY	Day of the year (1 - 366)
_HOUR	Hour (0 - 23)
_MINUTE	Minute (0 - 59)
_MONTH	Month of the year (1 - 12)
_SECOND	Second (0 - 59)
_TIME	* Current time (hh:mm:ss)
_YEAR	Year (1980 - 2099)

Error codes

?	* Exit code, last external program
_?	* Exit code, last internal command

ERRORLEVEL	* Exit code, last external program
_SYSERR	* Last Windows NT error code

Compatibility

=	* Substitutes escape character
+	* Substitutes command separator

Additional Notes

These internal variables are often used in batch files and aliases to examine system resources and adjust to the current computer settings. You can examine the contents of any internal variable (except **%=** and **%+**) from the command line with a command like this:

```
[c:\] echo %variablename
```

Some variables return values based on information provided by your operating system. These variables will only return correct information if the operating system provides it.

On disk volumes which do not support long filenames, variables which return a path or file name will return their result in upper or lower case depending on the value of the SETDOS /U switch or the UpperCase directive in the *.INI* file. On volumes which do support long filenames, these variables will return names as they are stored on the disk and no case shifting will be performed. Returned filename values which include long filenames are **not** quoted automatically; you must add quotes yourself if they are required for your use of the variable value (see Argument Quoting).

Examples

You can use these variables in a wide variety of ways depending on your needs. Here are just a few examples.

Store the current date and time in a file, then save the output of a DIR command in the same file:

```
echo Directory as of %_date %_time > dirsave
dir >> dirsave
```

Set up a prompt for the primary shell which displays the time and current directory, and a different one for secondary shells which includes the shell level rather than the time (see PROMPT for details about setting the prompt). Also set different background colors for the two shells, without changing the foreground color. You might use a sequence like this in your *4START* file (see Automatic Batch Files):

```
iff %_shell==0 then
prompt $t $p$g
color %_fg on blue
else
prompt [$z] $p$g
color %_fg on cyan
endiff
```

? contains the exit code of the last external command.   Many programs return a "0" to indicate success and a non-zero value to signal an error.   However, not all programs return an exit code.   If no explicit exit code is returned, the value of **%?** is undefined.

_? contains the exit code of the last internal command. It is set to "0" if the command was successful, "1" if a usage error occurred, "2" if another command processor error or an operating system error occurred, or "3" if the command was interrupted by **Ctrl-C** or **Ctrl-Break**. You must use or save this value immediately, because it is set by every internal command.

= returns the current escape character. Use this variable, instead of the actual escape character, if you want your batch files and aliases to work regardless of how the escape character is defined. For example, if the escape character is a caret [**^**] (the default in 4NT) both of the commands below will send a form feed to the printer. However, if the escape character has been changed, the first command will send the string "^f" to the printer, while the second command will continue to work as intended.

```
      echos ^f > prn
      echos %=f > prn
```

+ returns the current command separator. Use this variable, instead of the actual command separator, if you want your batch files and aliases to work regardless of how the command separator is defined. For example, if the command separator is an ampersand [**&**] (the default in 4NT) both of the commands below will display "Hello" on one line and "world" on the next. However, if the command separator has been changed the first command will display "Hello & echo world", while the second command will continue to work as intended.

```
echo Hello & echo world
echo Hello %+ echo world
```

_4VER is the current 4NT version (for example, "3.0").   The current decimal character is used to separate the major and minor version numbers (see DecimalChar for details).

_ANSI is always "0" in 4NT.   (Windows NT doesn't support ANSI sequences except in DOS sessions.)

_BATCH is the current batch nesting level.   **I**t is "0" if no batch file is currently being processed.

_BATCHLINE is the current line number in the current batch file.   It is "-1" if no batch file is currently being processed.

_BATCHNAME is the full path and file name of the current batch file.   It is an empty string if no batch file is currently being processed.

_BG is a string containing the first three characters of the screen background color at the current cursor location (for example, "Bla").

_BOOT is the boot drive letter, without a colon.

_CI is the current cursor shape in insert mode, as a percentage (see SETDOS /S and the CursorIns directive).

_CO is the current cursor shape in overstrike mode, as a percentage (see SETDOS /S and the CursorOver directive).

_CODEPAGE is the current code page number.

_COLUMN is the current cursor column (for example, "0" for the left side of the screen).

_COLUMNS is the current number of screen columns (for example, "80").

_COUNTRY is the current country code.

_CPU is the CPU type:

386	i386
486	i486
586	Pentium
686	Pentium Pro

_CWD is the current working directory in the format *d:\pathname*.

_CWDS has the same value as CWD, except it ends the pathname with a backslash [\].

_CWP is the current working directory in the format \pathname.

_CWPS has the same value as CWP, except it ends the pathname with a backslash [\].

_DATE contains the current system date, in the format mm-dd-yy (U.S.), dd-mm-yy (Europe), or yy-mm-dd (Japan).

_DAY is the current day of the month (1 to 31).

_DISK is the current disk drive, without a colon (for example, "C").

_DNAME returns the name of the description file.   By default, the description file is called *DESCRIPT.ION*.   The name can be changed with the DescriptionName directive in *4NT.INI*, or the SETDOS /D command.

_DOS is the operating system and command processor type.   Each JP Software command processor returns a different value, as follows:

Product	**Returns**
4DOS	"DOS"
4OS2	"OS2"
4NT	"NT"
Take Command/16	"WIN"
Take Command/32	"WIN95" (under Windows 95)
	"WIN32" (under Windows NT)
Take Command for OS/2	"PM" (for OS/2 Presentation Manager)

This variable is useful if you have batch files running in more than one environment, and need to take different actions depending on the underlying operating environment or command processor.

_DOSVER is the current operating system version (for example, "4.0").   The current decimal separator is used to separate the major and minor revision numbers (see DecimalChar for details).

_DOW is the first three characters of the current day of the week ("Mon", "Tue", "Wed", etc.).

_DOWI is the current day of the week as an integer (1 = Sunday, 2 = Monday, etc.).

_DOY is the day of the year (1 to 366).

ERRORLEVEL contains the exit code of the last external command. This variable is equivalent to the **?** variable. It is included only for compatibility with *CMD.EXE*.

_FG is a string containing the first three letters of the screen foreground color at the current cursor position (for example, "Whi").

_HLOGFILE returns the name of the current history log file (or an empty string if LOG /H is OFF). See LOG for information on logging.

_HOUR is the current hour (0 - 23).

_KBHIT returns 1 if one or more keystrokes are waiting in the keyboard buffer, or 0 if the keyboard buffer is empty.

_LASTDISK is the last valid drive letter, without a colon.

_LOGFILE returns the name of the current log file (or an empty string if LOG is OFF). See LOG for information on logging.

_MINUTE is the current minute (0 - 59).

_MONTH is the current month of the year (1 to 12).

_MOUSE always returns "1" in 4NT.

_NDP is the coprocessor type:

0	no coprocessor is installed
387	80387, 80486DX, 80487, Pentium, or Pentium Pro

_PID is the current process ID number.

_PIPE returns "1" if the current process is running inside a pipe or "0" otherwise.

_ROW is the current cursor row (for example, "0" for the top of the screen).

_ROWS is the current number of screen rows (for example, "25").

_SECOND is the current second (0 - 59).

_SHELL is the current shell nesting level.   The primary shell is level "0", and each subsequent secondary shell increments the level by 1.

_SYSERR is the error code of the last operating system error.   You will need a technical or programmer's manual to understand these error values.

_TIME contains the current system time in the format hh:mm:ss. The separator character may vary depending upon your country information.

_TRANSIENT is "1" if the current shell is transient (started with a ***/C***, see Starting 4NT for details), or "0" otherwise.

_WINDIR returns the pathname of the Windows NT directory.

_WINSYSDIR returns the pathname of the Windows NT system directory.

_WINTITLE returns the title of the current window.

_WINVER returns the current Windows NT version number.   The current decimal character is used to separate the major and minor version numbers   (see DecimalChar for details).

_YEAR is the current year (1980 to 2099).

Variable Functions

Variable functions are like internal variables, but they take one or more arguments (which can be environment variables or even other variable functions) and they return a value.

The list below gives a one-line description of each function, and a cross-reference which selects a separate help topic on that function. A few of the variables are simple enough that the one-line description is sufficient, but in most cases you should check for any additional information in the cross-referenced explanation if you are not already familiar with a function. You can also obtain help on any function with a **HELP @functionname** command at the prompt.

See the discussion after the function list for additional information and examples.

The variable functions are:

System status

@DOSMEM[b|k|m]	Size of largest free memory block
@READSCR[row,col,len]	Read characters from the screen

Drives and devices

@CDROM[d:]	CD-ROM drive detection (0 or 1)
@DEVICE[name]	Character device detection
@DISKFREE[d:,b|k|m]	Free disk space
@DISKTOTAL[d:,b|k|m]	Total disk space
@DISKUSED[d:,b|k|m]	Used disk space
@FSTYPE[d:]	File system type (FAT, NTFS, HPFS, CDFS, etc.)
@LABEL[d:]	Volume label
@READY[d:]	Drive ready status (0 or 1)
@REMOTE[d:]	Remote (network) drive detection (0 or 1)
@REMOVABLE[d:]	Removable drive detection (0 or 1)

Files

@ALTNAME[filename]	FAT-compatible file name
@ATTRIB[filename,rhsda]	File attribute test (0 or 1)
@DESCRIPT[filename]	File description
@FILEAGE[filename]	File age (date and time)
@FILECLOSE[n]	Close a file
@FILEDATE[filename]	File date
@FILEOPEN[filename,mode]	Open a file
@FILEREAD[n [,length]]	Read data from a file
@FILES[filename]	Count files matching a wildcard
@FILESEEK[n,offset,start]	Move a file pointer
@FILESEEKL[n,line]	Move file pointer to a line number

@FILESIZE[filename,b|k|m]	Size of files matching a wildcard
@FILETIME[filename]	File time
@FILEWRITE[n,text]	Write next line to a file
@FILEWRITEB[n,length,string]	Write bytes to a file
@FINDCLOSE[filename]	Close the search handle opened by @FINDFIRST
@FINDFIRST[filename [,-nrhsda]]	Find first matching file
@FINDNEXT[[filename [,-nrhsda]]]	Find next matching file
@LINE[filename,n]	Read a random line from a file
@LINES[filename]	Count lines in a file
@SEARCH[filename]	Path search
@UNIQUE[d:\path]	Create file with unique name

File names

@EXPAND[filename [,-nrhsda]]	Names of all matching files and directories
@EXT[filename]	File extension
@FILENAME[filename]	File name and extension
@FULL[filename]	Full file name with path
@LFN[filename]	Long path and filename
@NAME[filename]	File name without path or extension
@PATH[filename]	File path without name
@SFN[filename]	Short path and filename
@TRUENAME[filename]	True, fully-expanded filename

Strings and characters

@ASCII[c]	Numeric ASCII value for a character
@CHAR[n]	Character value for numeric ASCII
@FORMAT[[-][x][.y],string]	Formats (justifies) a string
@INDEX[string1,string2]	Position of one string in another
@INSERT[n,string1,string2]	Insert one string into another
@INSTR[start,length,string]	Extract a substring
@LEFT[n,string]	Leftmost characters of a string
@LEN[string]	Length of a string
@LOWER[string]	Convert string to lower case
@REPEAT[c,n]	Repeat a character
@REPLACE[string1,string2,text]	Replace all occurrences of one string with another
@RIGHT[n,string]	Rightmost characters of a string
@STRIP[chars,string]	Remove characters from string
@SUBSTR[string,start,length]	Extract a substring
@TRIM[string]	Remove blanks from a string

@UPPER[string]	Convert string to upper case
@WILD[string1,string2]	Wildcard comparison
@WORD[["sep",]n,string]	Extract a word from a string
@WORDS[["sep",]string]	Counts number of words in a string

Numbers and arithmetic

@COMMA[n]	Inserts commas in a number
@CONVERT[input,output,value]	Base Conversion
@DEC[%var]	Decremented value of a variable
@EVAL[expression]	Arithmetic calculations
@INC[%var]	Incremented value of a variable
@INT[n]	Integer part of a number
@NUMERIC[string]	Test if a string is numeric
@RANDOM[min,max]	Generate a random integer

Dates and times

@DATE[mm-dd-yy]	Convert date to number of days
@DAY[mm-dd-yy]	Day of the month
@DOW[mm-dd-yy]	Day of the week
@DOWI[mm-dd-yy]	Numeric day of the week
@DOY[mm-dd-yy]	Numeric day of the year
@MAKEAGE[date[,time]]	Convert date/time to file date/time
@MAKEDATE[n]	Convert number of days to date
@MAKETIME[n]	Convert number of seconds to time
@MONTH[mm-dd-yy]	Month of the year
@TIME[hh:mm:ss]	Convert time to number of seconds
@YEAR[mm-dd-yy]	Year number (2 digits)

Utility

@ALIAS[name]	Value of an alias
@CLIP[n]	Line from the clipboard
@EXEC[command]	Execute a command
@EXECSTR[command]	Execute, return string
@IF[condition,true,false]	Evaluates a test condition
@INIREAD[filename,section,entry]	Entry from *.INI* file
@INIWRITE[filename,section,entry,string]	Create or update entry in *.INI* file
@REXX[expr]	Execute a REXX expression
@SELECT[file,t,l,b,r,title]	Menu selection
@TIMER[n]	Elapsed time of specified timer

Additional Notes

Like all environment variables, these variable functions must be preceded by a percent sign in normal use (%@EVAL, %@LEN, etc.). All variable functions must have square brackets enclosing their argument(s). The argument(s) to a variable function cannot exceed 255 characters in length for all arguments taken as a group.

Specific Functions and Arguments

Some variable functions, like @DISKFREE, are shown with "**b|k|m**" as one of their arguments. Those functions return a number of bytes, kilobytes, or megabytes based on the "**b|k|m**" argument:

b	return the number of bytes
K	return the number of kilobytes (bytes / 1,024)
k	return the number of thousands of bytes (bytes / 1,000)
M	return the number of megabytes (bytes / 1,048,576)
m	return the number of millions of bytes (bytes / 1,000,000)

You can include commas (or the "thousands separator" character for your system) in the results from a "**b|k|m**" function by appending a "**c**" to the argument. For example, to add commas to a "**b**" or number of bytes result, enter "**bc**" as the argument. To set the thousands separator see the ThousandsChar directive.

Functions which accept a date as an argument use the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value. Two-digit years between 80 and 99 are interpreted as 1980 - 1999; values between 00 and 79 are interpreted as 2000 - 2079.

Several functions return filenames or parts of filenames. On an HPFS, NTFS, or LFN drive the strings returned by these functions may contain whitespace or other special characters. To avoid problems which could be caused by these characters, quote the returned name before you pass it to other commands, for example (either of these methods would work):

```
set fname="%@findfirst[pro*.*]"
echo First PRO file contains:
type %fname
.....

set fname=%@findfirst[pro*.*]
echo First PRO file contains:
type "%fname"
.....
```

If you don't use the quotes in the SET or TYPE command in this example, TYPE will not interpret any whitespace or special characters in the name properly.

In variable functions which take a drive letter as an argument, like @DISKFREE or @READY, the drive letter **must** be followed by a colon. The function will not work properly if you use the drive letter without the colon.

The @FILEREAD, @FILEWRITE, @FILEWRITEB, FILESEEK, FILESEEKL, and @FILECLOSE functions allow you to access files based on their file handle. These functions should only be used with file handles returned by @FILEOPEN! If you use them with any other file handle you may damage other files opened by 4NT (or, in a secondary shell, the program which started 4NT), or hang your system.

Many functions return values based on information provided by your operating system. Such functions will only return correct information if the operating system provides it. For example, @READY will not return accurate results if your operating system does not provide correct disk drive status information to the command processor.

Examples

You can use variable functions in a wide variety of ways depending on your needs. We've included a few examples below to give you an idea of what's possible. For a more comprehensive set of examples, see the *EXAMPLES.BTM* file which came with 4NT.

To set the prompt to show the amount of free memory (see PROMPT for details on including variable functions in your prompt):

```
[c:\] prompt (%%@dosmem[K]K) $p$g
```

Set up a simple command-line calculator. The calculator is used with a command like CALC 3 * (4 + 5):

```
[c:\] alias calc `echo The answer is:  %@eval[%&]`
```

The following batch file uses variable functions to implement "once a day" execution of a group of commands. It works by constructing a 6-digit number "yymmdd" from today's date, and comparing that to a number of the same type stored in the file *C:\ONCEADAY.DAT*. If today's date is numerically larger than the saved date, and the time is after 6:00 AM, then the "once a day" commands are run, and today's date is saved in the file as the new date for comparison. Otherwise, no action is taken. You can make this file simpler using the %@DATE and %@TIME functions instead of using %@INSTR to extract substrings of the %_DATE and %_TIME variables; we used the approach shown to demonstrate the use of %@INSTR.

```
rem  Temporary variables used to shorten example lines:
rem    DD is _date, DY is yymmdd date, TM is _time
set dd=%_date
set dy=%@instr[6,2,%dd]%@instr[0,2,%dd]%@instr[3,2,%dd]
set lastdate=0
iff exist c:\onceaday.dat then
  set lastdate=%@line[onceaday.dat,0]
endiff
iff %dy gt %lastdate then
  set tm=%_time
  iff "%@instr[0,2,%tm]%@instr[3,2,%tm]" gt "0600" then
    rem Commands to be executed once a day go here
    echo %dy > c:\onceaday.dat
  endiff
endiff
```

@ALIAS[name]:   Returns the contents of the specified alias as a string, or a null string if the alias doesn't exist.   When manipulating strings returned by @ALIAS you may need to disable certain special characters with the SETDOS /X command.   Otherwise, command separators, redirection characters, and other similar "punctuation" in the alias may be interpreted as part of the current command, rather than part of a simple text string.

@ALTNAME[filename]: Returns the FAT-style ("8.3" format) filename for the specified file. If the **filename** is already in 8.3 format, returns the filename if the file exists or an empty string if it does not. @ALTNAME will also return the shortened pathname if you provide a *path* in place of the **filename**.

@ASCII[c]:   Returns the numeric value of the specified ASCII character as a string.   For example **%@ASCII[A]** returns 65.   You can put an escape character [**^**] before the actual character to process. This allows quotes and other special characters as the argument (*e.g.*, **%@ASCII[^`]**).

@ATTRIB[filename,[nrhsda[,*p*]]]:   Returns a "1" if the specified file has the matching attribute(s); otherwise returns a "0".   The attributes are:

N	Normal (no attributes set)
R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

The attributes (other than **N**) can be combined (for example **%@ATTRIB[*MYFILE*,HS]**).   You can prefix an attribute with "-" to mean "everything except files with this attribute."

Without the optional ***p*** as a third argument, **ATTRIB** will only return a 1 if all of the attributes match.   With the ***p***, **ATTRIB** will return a 1 if there is a partial match.   For example, if *MYFILE.DAT* has R, H, and A attributes set:

```
%@attrib[myfile.dat,r]        returns 0 because there is
                        not an exact match

%@attrib[myfile.dat,r,p]returns 1 because there is
                        a partial match
```

If you do not specify any attributes, @ATTRIB will return the attributes of the specified file in the format **RHSAD**, rather than a "0" or "1".   Attributes which are not set will be replaced with an underscore.   For example, if *SECURE.DAT* has the read-only, hidden, and archive attributes set, **%@ATTRIB[*SECURE.DAT*]** would return **RH_A_**.

@CDROM[d:]: Returns "1" if the drive is a CD-ROM or "0" otherwise.

@CHAR[n]: Returns the character corresponding to an ASCII numeric value. For example **%@CHAR[65]** returns A.

@CLIP[n]: Returns line **n** from the clipboard.   The first line is numbered 0.   "**EOC**" is returned for all line numbers beyond the end of the clipboard.

@COMMA[n]:   Inserts commas, or the "thousands separator" character for your system, into a numeric string.   To set the thousands separator, see the ThousandsChar directive.

@CONVERT[input, output, value]: Converts a numeric string (**value**) from one number base (**input**) to another (**output**). Valid bases range from 2 to 36. The ***value*** must be a positive number between 0 and 2**32-1 (2,147,483,647). No error is returned if **value** is outside that range. For example, to convert "1010101" from binary to decimal, use this syntax:

```
%@convert[2,10,1010101]
```

@DATE[mm-dd-yy]: Returns the number of days since January 1, 1980 for the specified date. See Variable Functions for details on the date format.

@DAY[mm-dd-yy]: Returns the numeric day of the month for the specified date. See Variable Functions for details on the date format.

@DEC[%var]: Returns the same value as @EVAL[%var - 1]. That is, it retrieves and decrements the value of a variable. The variable itself is not changed; to do so, use a command like this:

```
set var=%@dec[%var]
```

@DESCRIPT[filename]: Returns the file description for the specified **filename** (see DESCRIBE).

@DEVICE[name]: Returns "1" if the specified name is a character device (such as a printer or serial port), or "0" if not.

@DISKFREE[d:,b|k|m]: Returns the amount of free disk space on the specified drive.

@DISKTOTAL[d:,b|k|m]: Returns the total disk space on the specified drive.

@DISKUSED[d:,b|k|m]: Returns the amount of disk space in use by files and directories on the specified drive.

@DOSMEM[b|k|m]: Returns the amount of free physical memory.

@DOW[mm-dd-yy]: Returns the first three characters of the day of the week for the specified date ("Mon", "Tue", "Wed", etc.).

@DOWI[mm-dd-yy]: Returns the day of the week for the specified date as an integer (1 = Sunday, 2 = Monday, etc.).

@DOY[mm-dd-yy]: Returns the day of year for the specified date (1-366).

@EVAL[expression]: Evaluates an arithmetic **expression**. @EVAL supports addition (**+**), subtraction (**-**), multiplication (*****), division (**/**), integer division (\, returns the integer part of the quotient), modulo (**%%**), and integer exponentiation (******). The **expression** can contain environment variables and other variable functions. @EVAL also supports parentheses, commas, and decimals. Parentheses can be nested. @EVAL will strip leading and trailing zeros from the result. When evaluating expressions, ******, *****, **/**, \, and **%%** take precedence over **+** and **-**. For example, 3 + 4 * 2 will be interpreted as 3 + 8, not as 7 * 2. To change this order of evaluation, use parentheses to specify the order you want. Also see @DEC and @INC.

To insure that your @EVAL expressions are interpreted correctly, spaces should be placed on both sides of an operator (e.g. %@eval[20 %% 3 + 4]).

The maximum precision is 16 digits to the left of the decimal point and 8 digits to the right of the decimal point. You can alter the default precision to the right of the decimal point on the Options 2 page of the OPTION dialogs, or with the EvalMax and EvalMin *4NT.INI* directives, and with the SETDOS /Fcommand. You can alter the decimal character from the Options 1 page of the OPTION dialogs, with the DecimalChar directive, or the SETDOS /G command.

To ensure that your @EVAL expressions are interpreted correctly, spaces should be placed on both sides of each operator, for example:

```
%@eval[(20 %% 3) + 4]
```

You can alter the precision for a single evaluation with the construct **@EVAL[expression=x.y]**. The **x** value specifies the minimum decimal precision (*i.e.,* the minimum number of decimal places displayed); the **y** value sets the maximum decimal precision. You can use ***=x,y*** instead of ***=x.y*** if the comma is your decimal separator. If **x** is greater than **y**, it is ignored. You can specify either or both arguments, for example:

```
@eval[3/7=.4]           returns 0.4286
@eval[3/7=2]      returns 0.42857143
@eval[3/6=2.4]          returns 0.50
```

@EXEC[[@]command]: Execute the **command** and return the numeric exit code. The **command** can be an alias, internal command, external command, or *.BTM*, *.BAT*, or *.CMD* file. @EXEC is primarily intended for running a program from within the PROMPT. It is a "back door" entry into command processing and should be used with extreme caution. Incorrect or recursive use of @EXEC may hang your system. By default, @EXEC returns the result code from the command; if you preface the command name with an '**@**' then @EXEC will return an empty string.

@EXECSTR[command]: Runs the specified **command** and returns the first line written to STDOUT by that **command**. @EXECSTR is primarily intended for running a program from within the PROMPT. It is a "back door" entry into command processing and should be used with extreme caution. Incorrect or recursive use of @EXECSTR may hang your system.

@EXECSTR is useful for retrieving a result from an external utility — for example, if you have an external utility called *NETTIME.EXE* which retrieves the time of day from your network server and writes it to standard output, you could save it in an environment variable using a command like this:

```
[c:\] set server_time=%@execstr[d:\path\nettime.exe]
```

If the same utility returned a result properly formatted for the TIME command you could also use it to set the time on your system:

```
[c:\] time %@execstr[d:\path\nettime.exe]
```

@EXPAND[filename[,-nrhsda]]: Returns, on a single line, the names of all files and directories that match the **filename** specification, which may contain wildcards and include lists. Returns an empty string if no files match. If the file list is longer than the allowed command line length, it will be truncated without an error message.

The second argument, if included, defines the attributes of the files that will be included in the search. The attributes are:

N	Normal (no attributes set)
R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

The attributes (other than **N**) can be combined (for example **%@EXPAND[*MYFILE*,HS]**). You can prefix an attribute with "-" to mean "everything except files with this attribute."

If the attribute argument is not used, hidden files, system files, and directories will be excluded from the returned list; all other files which match the **filename** will be included.

@EXT[filename]: Returns the extension (up to 64 characters) from a **filename**, without a leading period. On HPFS, NTFS, and LFN drives, the extension can be up to 64 characters long. On traditional FAT drives it can be up to 3 characters long.

@FILEAGE[filename]: Returns the date and time of the file as a single numeric value. The number can be used to compare the relative ages of two or more files, but can not be used for date and time calculations as it is not returned in identifiable units. Also see @MAKEAGE.

@FILECLOSE[n]: Closes the file whose handle is **n**. You cannot close handles 0, 1 or 2. Returns "0" if the file closed OK or "-1" if an error occurred. Be sure to read the cautionary note about file functions under Variable Functions.

@FILEDATE[filename[,acw]]: Returns the date a file was last modified, in the default country format (mm-dd-yy for the US). The optional second argument selects which date field is returned for files on an LFN, HPFS, or NTFS drive: **a** means the last access date, **c** means the creation date, and **w** means the last modification (write) date, which is the default.

@FILENAME[filename]: Returns the name and extension of a file, without a path.

@FILEOPEN[filename, read | write | append, [b | t]]: Opens the file in the specified mode and returns the file handle as an integer. Returns "-1" if the file cannot be opened.

The optional third parameter controls whether the file is opened in binary mode (**b**) or text mode (**t**). Text mode (the default) should be used to read text using @FILEREAD **without** a "length" parameter, and to write text using @FILEWRITE. Binary mode should be used to read binary data with @FILEREAD **with** a "length" parameter, and to write binary data with @FILEWRITEB.

Be sure to read the cautionary note about file functions under Variable Functions.

@FILEOPEN can also open named pipes. The pipe name must begin with \\.*pipe*\. @FILEOPEN first tries to open an existing pipe; if that fails it tries to create a new pipe. Pipes are opened in blocking mode, duplex access, byte-read mode, and inheritable. For more information on named pipes see your Windows NT documentation.

@FILEREAD[n,[length]]: Reads data from the file whose handle is **n**. Returns "**EOF**" if you attempt to read past the end of the file. If **length** is not specified @FILEREAD will read until the next CR or LF (end of line) character. If **length** is specified, @FILEREAD will read **length** bytes regardless of any end of line characters.

If you plan to read text a line at a time, without using **length**, you should open the file in text mode. If you plan to read binary data using **length**, you should open the file in binary mode. See @FILEOPEN for details on opening the file in the proper mode.

Be sure to read the cautionary note about file functions under Variable Functions.

@FILES[filename [,-nrhsda]]: Returns the number of files that match the **filename** specification, which may contain wildcards and include lists. Returns an empty string if no files match. The **filename** must refer to a single directory; to check several directories, use @FILES once for each directory, and add the results together with @EVAL. The second argument, if included, defines the attributes of the files that will be included in the search. The attributes are:

N	Normal (no attributes set)
R	Read-only
H	Hidden
S	System
D	Subirectory
A	Archive

The attributes (other than **N**) can be combined (for example **%@FILES[*MYFILE*,HS]**). You can prefix an attribute with "-" to mean "everything except files with this attribute."

@FILESEEK[n,offset,start]: Moves the file pointer **offset** bytes in the file whose handle is **n**. Returns the new position of the pointer, in bytes from the start of the file. Set **start** to 0 to seek relative to the beginning of the file, 1 to seek relative to the current file pointer, or 2 to seek relative to the end of the file. The **offset** value may be negative (seek backward), positive (seek forward), or zero (return current position, but do not change it). Be sure to read the cautionary note about file functions under Variable Functions.

@FILESEEKL[n,line]:   Moves the file pointer to the specified **line** in the file whose handle is **n**.   The first line in the file is numbered 0.   Returns the new position of the pointer, in bytes from the start of the file. Be sure to read the cautionary note about file functions under Variable Functions.

@FILESEEKL must read each line of the file up to the target line in order to position the pointer, and will therefore cause significant delays if used in a long loop or on a large file.

@FILESIZE[filename,b|k|m[,a]]: Returns the size of a file, or "-1" if the file does not exist. If the **filename** includes wildcards or an include list, returns the combined size of all matching files.

The optional third argument **a** (allocated), if used, instructs @FILESIZE to return the amount of space allocated for the file(s) on the disk, rather than the amount of data in the file. Network drives and compressed drives may not always report allocated sizes accurately, depending on the way the network or disk compression software is implemented.

@FILETIME[filename[,acw]]: Returns the time a file was last modified, in hh:mm format. The optional second argument selects which time field is returned for files on an LFN, HPFS, or NTFS drive: **a** means the last access time, **c** means the creation time, and **w** means the last modification (write) time, which is the default. The last access time is always returned as 00:00 on LFN drives

@FILEWRITE[n,text]: Writes a line to the file whose handle is **n**. Returns the number of bytes written, or "-1" if an error occurred. ***n*** must be a handle returned by @FILEOPEN; or 1 (for standard output) or 2 (for standard error).

If you plan to write text a line at a time with @FILEWRITE, you should open the file in text mode (see @FILEOPEN). If you want to write binary data you should use @FILEWRITEB instead, and open the file in binary mode.

Be sure to read the cautionary note about file functions under Variable Functions.

@FILEWRITEB[n,length,string]: Writes the specified number of bytes from the **string** to the file whose handle is **n**. Returns the number of bytes written, or "-1" if an error occurred.

If you plan to write binary data with @FILEWRITEB you should open the file in binary mode (see @FILEOPEN). If you want to write text a line at a time you may want to use the @FILEWRITE function instead, and open the file in text mode.

Be sure to read the cautionary note about file functions under Variable Functions.

@FINDCLOSE[filename]: Signals the end of a @FINDFIRST / @FINDNEXT sequence. You must use this function to release the directory search handle used for @FINDFIRST / @FINDNEXT.

@FINDFIRST[filename [,-nrhsda]]: Returns the name of the first file that matches the filename, which may include wildcards. The second argument, if included, defines the attributes of the files that will be included in the search. Returns an empty string if no files match. The attributes are:

N	Normal (no attributes set)
R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

The attributes (other than **N**) can be combined (for example **%@FINDFIRST[*MYFILE*,HS]**). @FINDFIRST will only find a file if all of the attributes match. You can prefix an attribute with "-" to mean "everything except files with this attribute."

@FINDFIRST always skips the "." and ".." entries when processing directory names.

After @FINDFIRST or the last @FINDNEXT, you must use @FINDCLOSE to avoid running out of directory search handles.

@FINDNEXT[[filename [,-nrhsda]]]: Returns the name of the next file that matches the filename passed to @FINDFIRST. Returns an empty string when no more files match. @FINDNEXT should only be used after a successful call to @FINDFIRST. The first argument is included for compatibility with previous versions, but is ignored; it can be omitted if the second argument is not used (*e.g.* %@FINDNEXT[]). The second argument, if included, defines the attributes of the files that will be included in the search. The attributes are:

N	Normal (no attributes set)
R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

The attributes (other than **N**) can be combined (for example **%@FINDNEXT[*MYFILE*,HS]**). @FINDNEXT will only find a file if all of the attributes match. You can prefix an attribute with "-" to mean "everything except files with this attribute."

@FINDNEXT always skips the "." and ".." entries when processing directory names.

After @FINDFIRST or the last @FINDNEXT, you must use @FINDCLOSE to avoid running out of directory search handles.

See the notes under Variable Functions about quoting returned long filenames.

@FORMAT[[-][x][.y],string]: Reformats a **string**, truncating it or padding it with spaces as necessary. If you use the minus **[-]**, the string is left-justified; otherwise, it is right-justified. The **x** value is the minimum number of characters in the result. The **y** value is the maximum number of characters in the result. You can combine the options as necessary. For example:

```
%@format[12,JPSoftware] returns "  JPSoftware"
%@format[.3,JPSoftware]       returns "JPS"
```

@FSTYPE[d:]: Returns the file system type for the specified drive. @FSTYPE will return "FAT" for a DOS-compatible drive with a file allocation table, "HPFS" for a drive that uses OS/2's high performance file system, "NTFS" for a drive that uses Windows NT's file system, or "CDFS" for a CD-ROM drive. It may return other values if additional file systems have been installed.

@FULL[filename]: Returns the fully qualified path and filename of a file. See the notes under Variable Functions about quoting returned long filenames.

@IF[condition,true,false]: Evaluates the **condition** and returns a string based on the result. The condition can include any of the tests allowed in the IF command. If the condition is true, @IF returns the first result string; if it is false, @IF returns the second string. For example, **echo %@IF[2 == 2,Correct!,Oops!]** displays "Correct!"

@INC[%var]: Returns the same value as %@EVAL[**%var** + 1]. That is, it retrieves and increments the value of a variable. The variable itself is not changed; to do so, use a command like this:

```
set var=%@inc[%var]
```

@INDEX[string1,string2]:   Returns the position of **string2** within **string1**, or "-1" if **string2** is not found. The first position in **string1** is numbered 0.

@INIREAD[filename,section,entry]: Returns an **entry** from the specified *.INI* file or an empty string if the **entry** does not exist. For example:

```
        %@iniread[c:\4nt\4nt.ini,4nt,history]
```

returns the size of the command history if it is specified in the [4NT] **section** of *4NT.INI*. If you don't specify a path for the *.INI* file, @INIREAD will look in the *\WINNT* and *\WINNT\SYSTEM* directories.

@INIWRITE[filename,section,entry,string]: Creates or updates an entry in the specified *.INI* file. For example:

```
        %@iniwrite[c:\4nt\4nt.ini,4nt,history,2048]
```

sets the size of the command history to 2,048 bytes the next time 4NT is started. @INIWRITE returns "0" for success or "-1" for failure. If you don't specify a path for the *.INI* file, @INIWRITE will look in the *\WINNT* and *\WINNT\SYSTEM* directories.

@INSERT[n, string1, string2]:   Inserts **string1** into **string2** starting at position **n.**   The first position in the string is postion 0.   For example, **%@insert[1,arm,wing]** returns the string "warming".

@INSTR[start, length, string]: Returns a substring, starting at the position **start** and continuing for **length** characters. If the **length** is omitted, it will default to the remainder of the **string**. If the **length** is negative, the start is relative to the right side of the **string**. The first character in the **string** is numbered 0; if the **length** is negative, the last character is numbered 0.

For example, %@INSTR[0,2,%_TIME] gets the current time and extracts the hour; %@INSTR[1,-2,%_TIME] extracts the seconds. If the **string** includes commas, it must be quoted with double quotes [**"**] or back-quotes [`]. The quotes **do** count in calculating the position of the substring. @SUBSTR is an older version of the same function.

@INT[n]:   Returns the integer part of the number **n**.

@LABEL[d:]:   Returns the volume label of the specified disk drive.

@LEFT[n,string]: Returns the leftmost **n** characters from the **string**. If **n** is greater than the length of the **string**, returns the entire **string**. For example, **%@left[2,jpsoft]** returns the string "jp".

@LEN[string]: Returns the length of a string.

@LFN[filename]:   Returns the long filename for a short ("8.3") **filename**.   The **filename** may contain any valid filename element including drive letter, path, filename and extension; the entire name including all intermediate paths will be returned in long name format.

See the notes under Variable Functions about quoting returned long filenames.

@LINE[filename,n]: Returns line **n** from the specified file. The first line in the file is numbered 0. "**EOF**" is returned for all line numbers beyond the end of the file.

The @LINE function must read each line of the file to find the line you request, and will therefore cause significant delays if used in a long loop or on a large file. For a more effective method of processing each line of a file in sequence use the FOR command, or @FILEOPEN and a sequence of @FILEREADs.

You can retrieve input from standard input if you specify CON as the filename. If you are redirecting input to @LINE using this feature, you must use command grouping or the redirection will not work properly (you can pipe to @LINE without a command group; this restriction applies only to input redirection). For example:

```
        (echo %@line[con,0]) < myfile.dat
```

@LINES[filename]: Returns the line number of the last line in the file, or "-1" if the file is empty. The first line in the file is numbered 0, so (for example) @LINES will return 0 for a file containing one line. To get the actual number of lines, use %@INC[%@LINES[filename]]. @LINES must read each line of the file in order to count it, and will therefore cause significant delays if used in a long loop or on a large file.

@LOWER[string]: Returns the **string** converted to lower case.

@MAKEAGE[date[,time]]: Returns the **date** and **time** (if included) as a single value in the same format as @FILEAGE. @MAKEAGE can be used to compare the time stamp of a file with a specific date and time, for example:

```
      if %@fileage[myfile] lt %@makeage[1/1/85] echo OLD!
```

The value returned by @MAKEAGE can be used for comparisons, but can not be used for date and time calculations because it is not in identifiable units.

@MAKEDATE[n]: Returns a date (formatted according to the current country settings). **n** is the number of days since 1/1/80. This is the inverse of @DATE.

@MAKETIME[n]: Returns a time (formatted according to the current country settings). **n** is the number of seconds since midnight. This is the inverse of @TIME.

@MONTH[mm-dd-yy]: Returns the month number for the specified date (1-12). See Variable Functions for details on the date format.

@NAME[filename]: Returns the base name of a file, without the path or extension. See the notes under Variable Functions about quoting returned long filenames.

@NUMERIC[string]: Returns "1" if the argument is composed entirely of digits (0 to 9), signs (**+** or **-**), and the thousands and decimal separators. Otherwise, returns "0". If the ***string*** begins with a decimal separator it is not considered numeric unless the next character is a digit, and there are no more decimal separators within the string. For example, ".07" is numeric, but ".a" and ".07.01" are not.

@PATH[filename]: Returns the path from a **filename**, including the drive letter and a trailing backslash, but not including the base name or extension. See the notes under Variable Functions about quoting returned long filenames.

@RANDOM[min, max]: Returns a "pseudo-random" value between **min** and **max**, inclusive. **min**, **max**, and the returned value are all integers. The random number generator is initialized from the system clock the first time it is used after the command processor starts, so it will produce a different sequence of random numbers each time you use it.

@READSCR[row,col,length]: Returns the text displayed on the screen at the specified location. The upper left corner of the screen is location 0,0.

The **row** and **column** can be specified as an offset from the current cursor location by preceding either value with a [**+**] or [**-**]. For example,

```
        %@readscr[-2,+2,10]
```

returns 10 characters from the screen, starting 2 rows above and 2 columns to the right of the current cursor position.

You can also specify the row and column as offsets from the current cursor position. Begin the value with a plus sign [**+**] to read the screen the specified number of rows below (or columns to the right of) the current position, or with a minus sign [**-**] to read the screen above (or to the left of) the current position.

@READY[d:]: Returns "1" if the specified drive is ready; otherwise returns "0".

@REMOTE[d:]:   Returns "1" if the specified drive is a remote (network) drive; otherwise returns "0".

@REMOVABLE[d:]:   Returns "1" if the specified drive is removable (*e.g.*, a floppy disk or removable hard disk); otherwise returns "0".

@REPEAT[c,n]: Returns the character **c** repeated **n** times.

@REPLACE[string1, string2, text]:   Replaces all occurrences of **string1** in the **text** string with **string2**. For example, **%@replace[w,ch,warming]** returns the string "charming".   The search **is** case-sensitive.

@REXX[expr]: Calls the REXX interpreter to execute the **expr**ession. Returns the result string from REXX; if the REXX expression does not return a string, **@REXX** returns the REXX numeric result code. See REXX Support for more information.

@RIGHT[n,string]:   Returns the rightmost **n** characters from the **string**.   If **n** is greater than the length of the ***string***, returns the entire string.   For example, **%@right[4,jpsoft]** returns the string "soft."

@SEARCH[filename[,path]]: Searches for the **filename** using the PATH environment variable or the specified **path**, appending an extension if one isn't specified. Returns the fully-expanded name of the file including drive, path, base name, and extension, or an empty string if a matching file is not found. If wildcards are used in the filename, **@SEARCH** will search for the first file that matches the wildcard specification, and return the drive and path for that file plus the wildcard filename (*e.g.*, *E:\UTIL\*.COM*).

@SELECT[filename,top,left,bottom,right,title[,1]]:   Pops up a selection window with the lines from the specified file, allowing you to display menus or other selection lists from within a batch file.   You can move through the selection window with standard popup window navigation keystrokes, including character matching (to change the navigation keys see Key Mapping Directives).   @SELECT returns the text of the line the scrollbar is on if you press **Enter**, or an empty string if you press **Esc**.

The ***filename*** must be in quotes if it contains whitespace or special characters.   The file size is limited only by available memory.   To select from lines passed through input redirection or a pipe, use CON as the ***filename***.

If you use the optional ***1*** argument after the window title, the list will be sorted alphabetically.

@SFN[filename]: Returns the fully expanded short ("8.3") filename for a long **filename**. The **filename** may contain any valid filename element including drive letter, path, filename and extension; the entire name including all intermediate paths will be returned in short name format. The ***filename*** should **not** be quoted.

@STRIP[chars,string]: Removes the characters in **chars** from the ***string*** and returns the result. For example, %@STRIP[AaEe,All Good Men] returns "ll Good Mn". The test is case sensitive. To include a comma in the **chars** string, enclose the entire first argument in double quotes. @STRIP will remove the quotes before processing the **string**.

@SUBSTR[string,start,length]: This is an older version of @INSTR.   The **string** parameter is at the start of the @SUBSTR argument list, and therefore cannot contain commas (because any commas in the string would be taken as argument separators).   @INSTR, which has the **string** argument last, does not have this restriction.

@TIME[hh:mm:ss]:   Returns the number of seconds since midnight for the specified time.   The time must be in 24-hour format; "am" and "pm" cannot be used.

@TIMER[n]: Returns the current split time for a stopwatch started with the TIMER command. The value of **n** specifies the timer to read and can be 1, 2, or 3.

@TRIM[string]:   Returns the **string** with the leading and trailing white space (space and tab characters) removed.

@TRUENAME [*filename*]: Returns the true, fully-expanded name for a file. TRUENAME will "see through" a JOIN or SUBST. Wildcards may not be used in the filename. @TRUENAME can handle simple drive substitutions such as those created by JOIN, SUBST, or most network drive mappings. However, it may not be able to correctly determine the true name if you use "nested" JOIN or SUBST commands, or a network which does not report true names properly.

@UNIQUE[d:\path]: Creates a zero-length file with a unique name in the specified directory, and returns the full name and path. If no **path** is specified, the file will be created in the current directory. The file name will be FAT-compatible (8 character name and 3-character extension) regardless of the type of drive on which the file is created. This function allows you to create a temporary file without overwriting an existing file.

@UPPER[string]: Returns the **string** converted to upper case.

@WILD[string1,string2]: Performs a comparison of the two strings, and returns "1" if they match or "0" if they don't match. The second argument, **string2**, may contain wildcards or extended wildcards; the first argument, **string1**, may not. The test is not case sensitive. The following example tests whether the \UTIL directory (or any directory that begins with the characters UTIL) is included in the PATH:

```
if %@wild[%path,*\UTIL*] == 1 command
```

@WORD[["xxx",]n,string]:   Returns the **n**th word in a **string**.   The first word is numbered 0.   If **n** is negative, words are returned from the end of the **string**.

You can use the first argument, **"xxx"** to specify the delimiters that you wish to use.   If you want to use a double quote as a delimiter, prefix it with an escape character.   If you don't specify a list of delimiters, @WORD will consider only spaces, tabs, and commas as word separators.   If the **string** argument is enclosed in quotation marks, you **must** enter a list of delimiters.   For example:

```
%@WORD[2,NOW IS THE TIME]      returns "THE"
%@WORD[-0,NOW IS THE TIME]     returns "TIME"
%@WORD[-2,NOW IS THE TIME]     returns "IS"
%@WORD["=",1,2 + 2=4]          returns "4"
```

@WORDS[["xxx"],string]: Returns the number of words in the **string**. You can use the first argument, **"xxx"** to specify the delimiters that you wish to use. If you want to use a double quote as a delimiter, prefix it with an escape character. If you don't specify a list of delimiters, @WORDS will consider only spaces, tabs, and commas as word separators. If the **string** argument is enclosed in quotation marks, you **must** enter a list of delimiters.

@YEAR[mm-dd-yy]:   Returns the year for the specified date.   See Variable Functions for details on the date format.   The year can be specified as two digits or four digits; @YEAR returns the same number of digits included in its argument.

4NT.INI

Part of the power of 4NT is its flexibility. You can alter its configuration to match your style of computing. Most of the configuration of 4NT is controlled through a file of initialization information called *4NT.INI*, which is discussed in the following sections:

Modifying the *.INI* File

Using the *.INI* File

.INI File Sections

.INI File Directives

.INI File Examples

We also discuss many ways of configuring 4NT in other sections of the help, for example:

» With aliases you can set default options for internal commands and create new commands.

» With executable extensions you can associate data files with the applications you use to open them.

» With the FILECOMPLETION environment variable and the FileCompletion *.INI* directive you can customize filename completion to match the command you are working with.

» With the COLORDIR environment variable and the ColorDir *.INI* directive you can set the colors used by the DIR and SELECT commands.

» With the SETDOS command, you can change some aspects of the command processor's operation "on the fly."

Modifying the .INI File

You can create, add to, and modify the *.INI* file in 2 ways: with the OPTION command and by editing the file with any ASCII editor. OPTION displays a set of dialogs which allow you to modify the settings that are used most often. When you exit from the dialogs, you can select the **Save** button to save your changes in the *.INI* file for use in the current session and all future sessions, select the **Use** or **OK** button to use your changes in the current session only, or discard the changes you have made by selecting the **Cancel** button. See the OPTION command for additional details.

Changes you make in the **Startup** section of the OPTION dialogs will only take effect when you restart the session or window in which 4NT is running.

OPTION handles most standard *.INI* file settings. A few more advanced settings, as well as all settings that affect the interpretation of keystrokes, cannot be modified with OPTION and must be inserted or modified manually. For more details see the OPTION command.

You can also create, add to, and edit the *4NT.INI* file "manually" with any ASCII text editor. 4NT reads the *.INI* file when it starts, and configures itself accordingly. The *.INI* file is not re-read when you change it manually. For manual changes to take effect, you must restart the session or window in which 4NT is running. If you edit the *.INI* file manually, make sure you save the file in ASCII format.

Each item that you can include in the *.INI* file has a default value. You only need to include entries in the file for settings that you want to change from their default values.

Format

Most lines in the *.INI* file consist of a one-word **directive**, an equal sign [**=**], and a **value**. For example, in the following line, the word "Environment" is the directive and "2048" is the value:

```
Environment = 2048
```

Any spaces before or after the equal sign are ignored.

If you have a long string to enter in the *.INI* file (for example, for the ColorDir directive), you must enter it all on one line. Strings cannot be "continued" to a second line. Each line may be up to 1023 characters long.

The format of the value part of a directive line depends on the individual directive. It may be a numeric value, a single character, a choice (like "Yes" or "No"), a color setting, a key name, a path, a filename, or a text string. The value begins with the first non-blank character after the equal sign and ends at the end of the line or the beginning of a comment.

Blank lines are ignored in the *.INI* file and can be used to separate groups of directives. You can place comments in the file by beginning a line with a semicolon [**;**]. You can also place comments at the end of any line except one containing a text string value. To do so, enter at least one space or tab after the value, a semicolon, and your comment, like this:

```
Environment = 2048      ;set standard environment size
```

If you try to place a comment at the end of a string value, the comment will become part of the string and will probably cause an error.

If you use the OPTION dialogs to modify the *.INI* file, comments on lines modified from within the dialogs will not be preserved when the new lines are saved. To be sure *.INI* file comments are preserved, put them on separate lines in the file.

If you want to include the text of one *.INI* file within another (for example, if you have a set of common directives used by several JP Software products), see the Include directive.

Using the .INI File

Some settings in the *.INI* file are initialized when you install 4NT, so you will probably have a *4NT.INI* file even if you didn't create one yourself.

4NT primary shells search for the *.INI* file in three places:

» If there is an "@d:\path\inifile" option on the startup command line, the command processor will use the path and file name specified there, and will not look elsewhere. See Starting 4NT for more details.

» If there is no *.INI* file name on the startup command line, the search proceeds to the same directory where the command processor program file (*4NT.EXE*) is stored. This is the "normal" location for the *.INI* file. 4NT determines this directory automatically.

» If the *.INI* file is not found in the directory where the program file is stored, a final check is made in the root directory of the boot drive.

When 4NT is loaded as a secondary shell, it does not search for the *.INI* file. Instead, it retrieves the primary shell's *.INI* file data, processes the **[Secondary]** section of the original *.INI* file if necessary (see *.INI* File Sections), and then processes any "@d:\path\inifile" option on the secondary shell command line (see Starting 4NT). You can override this behavior with the NextINIFile directive.

Secondary shells automatically inherit the configuration settings currently in effect in the previous shell. If values have been changed by SETDOS since the primary shell started, the current values will be passed to the secondary shell. If the previous shell's *.INI* file had a **[Secondary]** section, it will then be read and processed (see *.INI* File Sections). If not, the previous shell's settings will remain in effect.

For example, you might set BatchEcho to Yes in the *.INI* file, to enable batch file echo. If you then use SETDOS /V0 to turn off batch file echoing in the primary shell, then any secondary shells will inherit the SETDOS setting, rather than the original value from the *.INI* file; *i.e.*, batch files in the secondary shell will default to no echo.

If you want to force secondary shells to start with a specific value for a particular directive, regardless of any changes made with SETDOS in a previous shell, repeat the directive in the **[Secondary]** section of the *.INI* file.

The SETDOS command can override several of the *.INI* file directives. For example, the cursor shape used by 4NT can be adjusted either with the CursorIns and CursorOver directives or the SETDOS /S command. The correspondence between SETDOS options and *.INI* directives is noted with each directive, and under each option of the SETDOS command.

When the command processor detects an error while processing the *.INI* file, it displays an error message and prompts you to press a key to continue processing the file. This allows you to note any errors before the startup process continues. The directive in error will retain its previous or default value. Only the most catastrophic errors (like a disk read failure) will terminate processing of the remainder of the *.INI* file. If you don't want a pause after each error, use a "PauseOnError = No" directive at the beginning of the *.INI* file.

If you need to test different values for an *.INI* directive without repeatedly editing the *.INI* file, use the OPTION command or see the INIQuery directive.

.INI File Sections

The *.INI* file has three possible sections: the first or **global** section, named **[4NT]**; the **[Primary]** section; and the **[Secondary]** section.   Each section is identified by the section name in square brackets on a line by itself.

Directives in the global section are effective in all shells.   In most cases, this is the only section you will need.   Any changes you make to the *.INI* file with the OPTION command are stored in the global section.

The **[Primary]** and **[Secondary]** sections include directives that are used only in primary and secondary shells, respectively.   You don't need to set up these sections unless you want different directives for primary and secondary shells.

Directives in the **[Primary]** section are used for the first or primary shell.   The values are passed automatically to all secondary shells, unless overridden by a directive with the same name in the **[Secondary]** section.

Directives in the **[Secondary]** section are used in secondary shells only, and override any corresponding primary shell settings.   For example, these lines in the *.INI* file:

```
[Primary]
ScreenRows = 25
[Secondary]
ScreenRows = 50
```

mean to assume that you have 25 rows on the screen in the primary shell and 50 lines in all secondary shells.

Sections that begin with any name other than **[4NT]**, **[Primary]**, or **[Secondary]** are ignored.

.INI File Directives

For information on specific directives see the separate topic for each type of directive:

- Initialization Directives
- Configuration Directives
- Color Directives
- Key Mapping Directives
- Advanced Directives

These topics list the directives, with a one-line description of each, and a cross-reference which selects a full screen help topic on that directive. A few of the directives are simple enough that the one-line description is sufficient, but in most cases you should check for any additional information in the cross-reference topic if you are not already familiar with the directive.

You can also obtain help on most directives with a **HELP directive** command at the prompt.

There are 8 types of directives in the *.INI* file. The different types of directives are shown in the descriptions as follows:

» **Name = nnnn (1234)**: This directive takes a numeric value which replaces the "nnnn." The default value is shown in parentheses.

» **Name = c (X)**: This directive accepts a single character as its value. The default character is shown in parentheses. You must type in the actual character; you cannot use a key name.

» **Name = CHOICE1 | Choice2 | ...** : This directive takes a choice value. The possible choices are listed, separated by vertical bars. The default value is shown in all upper case letters in the directive description, but in your file any of the choices can be entered in upper case or lower case. For example, if the choices were shown as "YES | No" then "YES" is the default.

» **Name = Color**: This directive takes a color specification. See Colors and Color Names.

» **Name = Key (Default)**: This directive takes a key specification. See Key Names for the format of key names.

» **Name = Path**: This directive takes a path specification, but not a filename. The value should include both a drive and path (*e.g.*, *C:\4NT*) to avoid any possible ambiguities. A trailing backslash [\] at the end of the path name is acceptable but not required. Any default path is described in the text.

» **Name = File**: This directive takes a filename. We recommend that you use a full filename including the drive letter and path to avoid any possible ambiguities. Any default filename is described in the text.

» **Name = String**: This directive takes a string in the format shown. The text describes the default value and any additional requirements for formatting the string correctly. **No comments are allowed**.

4NT contains a fixed-length area for storing strings entered in the *.INI* file, including file names, paths, and other strings. This area is large and is unlikely to overflow; if it does, you will receive an error message. If this occurs, reduce the complexity of your *.INI* file or contact our technical support department for assistance.

.INI File Examples

This example for 4NT configures certain special characters to match 4DOS, and changes other default settings to suit the user's preferences.

```
[4NT]
InstallPath = c:\4nt300              ;installation directory
PauseOnError = No                    ;don't stop on INI errors
CommandSep = ^                       ;4DOS command separator
EscapeChar =                         ;4DOS escape character
ParameterChar = &                    ;4DOS parameter character
BatchEcho = No                       ;default to ECHO OFF
History = 2048                       ;expand history to 2K bytes
BeepFreq = 880                       ;make beep higher pitch
EditMode = Insert                    ;insert mode for cmd edit
CursorOver = 100                     ;overstrike cursor 100%
CursorIns = 10                       ;insert cursor 10%
ListFind = F5                        ;F5 does a find in LIST
ListNext = F6                        ;and F6 does a find next
StdColors = bri cya on blu           ;default colors
ListColors = bri whi on blu          ;colors for LIST
SelectColors = bri whi on blu        ;same colors for SELECT
colordir = DIRS:bri yel;com exe bat btm cmd:bri whi
```

Initialization Directives

The directives in this section control how 4NT starts and where it looks for its files. The initialization directives are:

4StartPath	Path for 4START and 4EXIT
DirHistory	Size of directory history list
DuplicateBugs	Duplicate bugs in *CMD.EXE*
History	Size of history list
INIQuery	Query for each line in *4NT.INI*
LoadAssociations	Controls loading of Windows' file associations
LocalAliases	Local vs. global aliases
LocalDirHistory	Local vs. global directory history
LocalHistory	Local vs. global history
PauseOnError	Pause on errors in *4NT.INI*
TreePath	Path for directory database
WindowState	Initial state for the 4NT window
WindowX, WindowY, WindowWidth, WindowHeight	Initial size and position of the 4NT window

4StartPath = Path: Sets the drive and directory where the *4START* and *4EXIT* batch files (if any) are located.

DirHistory = nnnn (256): Sets the amount of memory allocated to the directory history in bytes. The allowable range of values is 256 to 32767 bytes. If you use a global directory history list, the DirHistory value is ignored in all shells except the shell which first establishes the global list.

DuplicateBugs = YES | No: Tells the 4NT parser to duplicate certain well-known bugs in *CMD.EXE*. The only bug currently replicated is in the IF command.

History = nnnn (1024): Sets the amount of memory allocated to the command history list in bytes. The allowable range of values is 256 to 32767 bytes. If you use a global history list (see Command History and Recall), the History value is ignored in all shells except the shell which first establishes the global list.

INIQuery = Yes | NO: If set to **Yes**, a prompt will be displayed before execution of each subsequent line in the current *.INI* file. This allows you to modify certain directives when you start 4NT in order to test different configurations. INIQuery can be reset to **No** at any point in the file. Normally INIQuery = Yes is only used during testing of other *.INI* file directives.

The prompt generated by INIQuery = Yes is:

```
      [contents of the line]  (Y/N/Q/R/E)  ?
```

At this prompt, you may enter:

Y = Yes:	Process this line and go on to the next.
N = No:	Skip this line and go on to the next.
Q = Quit:	Skip this line and all subsequent lines.
R = Rest:	Execute this and all subsequent lines.
E = Edit:	Edit the value for this entry.

If you choose E for Edit, you can enter a new value for the directive, but not a new directive name.

LoadAssociations = YES **|** No: **No** prevents 4NT from loading Windows' direct file associations from the Windows Registry for use when searching for executable files.   The default of **Yes** allows loading of the file associations.   See Windows File Associations for additional details.

LocalAliases = Yes | NO: **No** forces all copies of 4NT to share the same alias list.   **Yes** keeps the lists for each shell separate.   See ALIAS for more details on local and global alias lists.

LocalDirHistory = Yes | NO: **No** forces all copies of the command processor to share the same directory history.   **Yes** keeps the directory histories for each shell separate.   See Directory History Window for more details on local and global directory histories.

LocalHistory = Yes | NO: **No** forces all copies of 4NT to share the same history list. **Yes** keeps the lists for each shell separate. See Command History and Recall for more details on local and global history lists.

PauseOnError = YES | No: **Yes** forces a pause with the message "Error in *filename*, press any key to continue processing" after displaying any error message related to a specific line in the *.INI* file. **No** continues processing with no pause after an error message is displayed.

TreePath = Path: Sets the location of *JPSTREE.IDX*, the file used for extended directory searches. By default, the file is placed in the root directory of drive C:\.

WindowState = STANDARD | Maximize | Minimize | Custom: Sets the initial state of the 4NT window. **Standard** puts the window in the default position on the Windows NT desktop, and is the default setting. **Maximize** maximizes the window; **Minimize** minimizes it. **Custom** uses the position and size specified by WindowX, etc. If you use Maximize, Minimize, or Custom, you may see the 4NT window appear briefly in the Standard position as it is created by Windows NT, then switch to the new state.

WindowX = nnnn, **WindowY** = nnnn, **WindowWidth** = nnnn, **WindowHeight** = nnnn: These 4 directives set the initial size and position of the 4NT window.   They are ignored unless WindowState is set to Custom.   The measurements are in pixels or pels.   **WindowX** and **WindowY** refer to the position of the top left corner of the window relative to the top left corner of the screen.

Configuration Directives

These directives control the way that 4NT operate.   Some can be changed with the SETDOS command while 4NT is running. Any corresponding SETDOS command is listed in the description of each directive. The configuration directives are:

AmPm	Time display format
AppendToDir	"\" on directory names in filename completion
BatchEcho	Default batch file echo state
BeepFreq	Default beep frequency
BeepLength	Default beep length
CDDWinLeft, CDDWinTop, CDDWinWidth, CDDWinHeight	Initial position and size of the directory change window
CommandSep	Multiple command separator character
CursorIns	Cursor shape in insert mode
CursorOver	Cursor shape in overstrike mode
DecimalChar	Decimal separator
DescriptionMax	Maximum length of file descriptions
DescriptionName	Name of file to hold file descriptions
Descriptions	Enable / disable description processing
EditMode	Editing mode (insert / overstrike)
EscapeChar	4NT escape character
EvalMax	Max digits after decimal point in @EVAL
EvalMin	Min digits after decimal point in @EVAL
ExecWait	Forces 4NT to wait for external programs to complete
FileCompletion	Filename completion by extension
FuzzyCD	Selects Extended Directory Search mode
HistCopy	History copy mode
HistLogName	History log file name
HistMin	Minimum command length to save
HistMove	History move mode
HistWrap	History wrap mode
LogName	Log file name
NoClobber	Overwrite protection for output redirection
ParameterChar	Alias / batch file parameter character
PopupWinLeft, PopupWinTop, PopupWinWidth, PopupWinHeight	Initial position and size of popup windows
Printer	LIST print device
ScreenRows	Screen height
TabStops	Tab width in LIST

ThousandsChar	Thousands separator
UpperCase	Force file names to upper case

AmPm = Yes | NO | Auto: **Yes** displays times in 12-hour format with a trailing "a" for AM or "p" for PM. The default of **No** forces a display in 24-hour time format. **Auto** formats the time according to the country code set for your system. AmPm controls the time displays used by DIR and SELECT, in LOG files, and the output of the TIMER, DATE, and TIME commands. It has no effect on %_TIME, %@MAKETIME, the $t and $T options of PROMPT, or date and time ranges.

AppendToDir = Yes | NO: **Yes** appends a trailing "\" to directory names when doing filename completion.   The default is **No**.   (Regardless of the setting of this directive, a trailing backslash is always appended to a directory name at the beginning of the command line to enable automatic directory changes.)

BatchEcho = YES | No: Sets the default batch echo mode. **Yes** enables echoing of all batch file commands unless ECHO is explicitly set off in the batch file. **No** disables batch file echoing unless ECHO is explicitly set on. Also see SETDOS /V.

BeepFreq = nnnn (440): Sets the default BEEP command frequency in Hz. This is also the frequency for "error" beeps (for example, if you press an illegal key). To disable all error beeps set this or BeepLength to 0. If you do, the BEEP command will still be operable, but will not produce sound unless you explicitly specify the frequency and duration.

BeepLength = nnnn (2):   Sets the default BEEP length in system clock ticks (approximately 1/18 of a second per tick). BeepLength is also the default length for "error" beeps (for example, if you press an illegal key).

CDDWinLeft = nnnn, **CDDWinTop** = nnnn, **CDDWinWidth** = nnnn, **CDDWinHeight** = nnnn: These values set the position and size of the popup window used by extended directory searches, in characters, including the border. The defaults are 3, 3, 72, and 16, respectively (*i.e.*, a window beginning in column 3, row 3, 72 columns wide and 16 rows high). The position is relative to the top left corner of the screen. The width and height values include the space required for the window border. The window cannot be smaller than than 10 columns wide by 5 rows high (including the border). The values you enter will be adjusted if necessary to keep a minimum-size window visible on the screen.

The window is normally displayed with a shadow, but if you specify a window starting at column 0 and extending to the right margin, the shadow is eliminated; this may be useful to prevent speech software from reading text in the shadow area while viewing the window.

CommandSep = c: This is the character used to separate multiple commands on the same line. The default is the ampersand [**&**]. You cannot use any of the redirection characters (**| > <**) or any of the whitespace characters (space, tab, comma, or equal sign). The command separator is saved by SETLOCAL and restored by ENDLOCAL.

Also see SETDOS /C, the %+ internal variable, and Special Character Compatibility for information on using compatible command separators for two or more products.

CursorIns = nnnn (100):   This is the shape of the cursor for insert mode during command-line editing and all commands which accept line input (DESCRIBE, ESET, etc.).   The size is a percentage of the total character cell size, between 0% and 100%.   If **CursorIns** or CursorOver is set to -1, the command processor will not attempt to modify the cursor shape at all; you can use this feature to give another program full control of the cursor shape.   Because of the way video drivers map the cursor shape, you may not get a smooth progression in cursor shapes as CursorIns and CursorOver change.

Also see SETDOS /S.

CursorOver = nnnn (15): This is the shape of the cursor for overstrike mode during command-line editing and all commands which accept line input. The size is a percentage of the total character cell size, between 0% and 100%. If **CursorOver** or CursorIns is set to -1, the command processor will not attempt to modify the cursor shape at all; you can use this feature to give another program full control of the cursor shape.

Also see SETDOS /S.

DecimalChar = . | , | AUTO: Sets the character used as the decimal separator for @EVAL, numeric IF and IFF tests, version numbers, and other similar uses. The only valid settings are period [**.**], comma [**,**], and **Auto** (the default). A setting of Auto tells the command processor to use the decimal separator associated with your current country code. If you change the decimal character you must also adjust the thousands character (with ThousandsChar) so that the two characters are different. Also see SETDOS /G.

DescriptionMax = nnnn (40): Controls the description length limit for DESCRIBE. The allowable range is 20 to 511 characters.

DescriptionName = File: Sets the name of the hidden file in each directory that will hold file descriptions. If you don't use this directive, the description files will be named *DESCRIPT.ION*. Use this directive with caution, because changing the name from the default will make it difficult to transfer file descriptions to another system. Also see SETDOS /D.

Descriptions = YES | No: Turns description handling on or off during the file processing commands COPY, DEL, MOVE, and REN. If set to **No**, 4NT will not update the description file when files are moved, copied, deleted or renamed. Also see SETDOS /D.

EditMode = Insert | OVERSTRIKE: This directive lets you start the command-line editor in either insert or overstrike mode.   Also see SETDOS /M.

EscapeChar = c: Sets the character used to suppress the normal meaning of the following character. The default is a caret [**^**]. See Escape Character for a description of special escape sequences. You cannot use any of the redirection characters (**|**, **>**, or **<**) or the whitespace characters (space, tab, comma, or equal sign) as the escape character. The escape character is saved by SETLOCAL and restored by ENDLOCAL.

Also see SETDOS /E, the %= internal variable, and Special Character Compatibility for information on using compatible escape characters for two or more products.

EvalMax = nnnn (0): Controls the maximum number of digits after the decimal point in values returned by @EVAL. The allowable range is 0 to 8. This directive will be ignored if EvalMin is larger than EvalMax. This setting can be overridden with the construct @EVAL[expression=n.n].

Also see SETDOS /F.

EvalMin = nnnn (0): Controls the minimum number of digits after the decimal point in values returned by @EVAL. The allowable range is 0 to 8. This directive will be ignored if EvalMin is larger than EvalMax. This setting can be overridden with the construct @EVAL[expression=n.n].

Also see SETDOS /F.

ExecWait = Yes | NO: Controls whether 4NT waits for an external program to complete before redisplaying the prompt. See Waiting for Applications to Finish for details on the effects of this directive.

FileCompletion = cmd1: ext1 ext2 ...; cmd2: ext3 ext4 ...   Sets the files made available during filename completion for selected commands.   The format is the same as that used for the FILECOMPLETION environment variable.   See Filename Completion for a detailed explanation of selective filename completion.

FuzzyCD = 0 **|** 1 **|** 2 **|** 3:   Enables or disables extended directory searches, and controls their behavior.   A setting of **0** (the default) disables extended searches.   For complete details on the meaning of the other settings see Extended Directory Searches.

HistCopy = Yes | NO: Controls what happens when you re-execute a line from the command history. If this option is set to **Yes**, the line is appended to the end of the history list. By default, or if this option is set to **No**, the command is not copied. The original copy of the command is retained at its original position in the list regardless of the setting of HistCopy.

Set this option to **No** if you want to use HistMove = Yes; otherwise, the HistCopy setting will override HistMove.

HistLogName = File: Sets the history log file name and path.   Using HistLogName does not turn history logging on; you must use a LOG /H ON command to do so.

HistMin = nnnn (0): Sets the minimum command-line size to save in the command history list. Any command line whose length is less than this value will not be saved. Legal values range from 0, which saves everything, to 1024, which disables all command history saves.

HistMove = Yes | NO: If set to Yes, a recalled line is moved to the end of the command history. The difference between this directive and HistCopy is that HistCopy = Yes copies each recalled line to the end of the history but leaves the original in place. HistMove = Yes places the line at the end of history and removes the original line. This directive has no effect if HistCopy = Yes.

HistWrap = YES | No: Controls whether the command history recall "wraps" when you reach the top or bottom of the list. The default setting enables wrapping, so the list appears "circular". If HistWrap is set to No, history recall will stop at the beginning and end of the list rather than wrapping. This setting affects history recall at the prompt only; the command history window never wraps.

LogName = File: Sets the log file name and path. If only a path is given, the default log file name (*4NTLOG*) will be used.   Using LogName does not turn logging on; you must use a LOG ON command to do so.

NoClobber = Yes | NO: If set to **Yes**, will prevent standard output redirection from overwriting an existing file, and will require that the output file already exist for append redirection. Also see SETDOS /N.

ParameterChar = c: Sets the character used after a percent sign to specify all or all remaining command-line arguments in a batch file or alias (*e.g.*, **%&** or **%n&**; see Batch Files and ALIAS).   The default is the dollar sign [**$**].   The parameter character is saved by SETLOCAL and restored by ENDLOCAL.

Also see SETDOS /P.   See Special Character Compatibility for information on using compatible parameter characters for two or more products..

PopupWinLeft = nnnn, **PopupWinTop** = nnnn, **PopupWinWidth** = nnnn, **PopupWinHeight** = nnnn: These values set the position and size of the command-line, directory history, and filename completion windows, and most other popup windows (see CDDWinLeft etc. for the extended directory search window). The values are in characters, and include the border. The defaults are 40, 1, 36, and 12, respectively (*i.e.*, a window beginning in column 40, row 1, 36 columns wide and 12 rows high). The position is relative to the top left corner of the screen. The width and height values include the space required for the window border. The window cannot be smaller than than 10 columns wide by 5 rows high (including the border). The values you enter will be adjusted if necessary to keep a minimum-size window visible on the screen.

The window is normally displayed with a shadow, but if you specify a window starting at column 0 and extending to the right margin, the shadow is eliminated; this may be useful to prevent speech software from reading text in the shadow area while viewing the window.

Printer = devicename: Sets the output device that the LIST command will print to. By default, LPT1 is used. The device can be PRN, LPT1 to 3, COM1 to 4, NUL (which will disable printed output) or any other installed character device.

ScreenRows = nnnn: Sets the number of screen rows used by the video display.   Normally the screen size is determined automatically, but if you have a non-standard display you may need to set it explicitly. This value does not affect screen scrolling, which is controlled by Windows NT and your video driver. ScreenRows is used only by the LIST and SELECT commands, the paged output options of other commands (*e.g.*, TYPE /P), and error checking in the screen output commands.   Also see SETDOS /R.

TabStops = nnnn (8): Sets the tab stops for files displayed with the LIST command. Setting TabStops=3, for example, will place a tab stop in every third column. The allowable range is 1 to 32.

ThousandsChar = . | , | AUTO: Sets the character used as the thousands separator for numeric output. The only valid settings are period [**.**], comma [**,**], and **Auto** (the default). A setting of Auto tells the command processor to use the thousands separator associated with your current country code. If you change the thousands character you must also adjust the decimal character (with DecimalChar) so that the two characters are different. Also see SETDOS /G.

UpperCase = Yes | NO: **Yes** specifies that file and directory names should be displayed in the traditional upper-case by internal commands like COPY and DIR. **No** allows the normal 4NT lower-case style. This directive does not affect the display of filenames on drives which support long filenames (see File Names for additional details). Also see SETDOS /U.

Color Directives

These directives control the colors that 4NT use for its displays. For complete details on color names see Colors and Color Names. The color directives are:

CDDWinColors	Directory change window colors
ColorDir	Directory colors
InputColors	Input colors
ListboxBarColors	Light bar color in list boxes
ListColors	LIST display colors
ListStatBarColors	LIST status bar colors
PopupWinColors	Popup window colors
SelectColors	SELECT display colors
SelectStatBarColors	SELECT status bar colors
StdColors	Standard display colors

CDDWinColors = Color: Sets the default colors for the popup window used by extended directory searches.   If this directive is not used, the colors will be reversed from the current colors on the screen.

ColorDir = ext1 ext2 ...:colora;ext3 ext4 ... :colorb; ...: Sets the directory colors used by DIR and SELECT. The format is the same as that used for the COLORDIR environment variable.   See Color-Coded Directories for a detailed explanation.

InputColors = Color: Sets the colors used for command-line input. This setting is useful for making your input stand out from the normal output.

ListboxBarColors = Color: Sets the color for the highlight bar in the popup list boxes (*i.e.*, command history window, filename completion window, @SELECT window, etc.).

ListColors = Color: Sets the colors used by the LIST command. If this directive is not used, LIST will use the current default colors set by the CLS or COLOR command or by the StdColors directive.

ListStatBarColors = Color: Sets the colors used on the LIST status bar.   If this directive is not used, LIST will set the status bar to the reverse of the screen color (the screen color is controlled by ListColors).

PopupWinColors = Color: Sets the default colors for the command-line, directory history, and filename completion windows, and most other popup windows (see CDDWinColors for the extended directory search window). If this directive is not used, the colors will be reversed from the current colors on the screen.

SelectColors = Color: Sets the color used by the SELECT command. If this directive is not used, SELECT will use the current default colors set by the CLS or COLOR command or by the StdColors directive.

SelectStatBarColors = Color: Sets the color used on the SELECT status bar. If this directive is not used, SELECT will set the status bar to the reverse of the screen color (the screen color is controlled by SelectColors).

StdColors = Color: Sets the standard colors to be used when CLS is used without a color specification, and for LIST and SELECT if ListColors and SelectColors are not used.   Using this directive is similar to placing a COLOR command in *4START.BAT*.   StdColors takes effect the first time CLS, LIST, or SELECT is used after 4NT starts, but will not affect the color of error or other messages displayed during the loading and initialization process.

Key Mapping Directives

These directives allow you to change the keys used for command-line editing and other internal functions. They are divided into four types, depending on the context in which the keys are used. For a discussion and list of directives for each type see:

General Input Keys

Command-Line Editing Keys

Popup Window Keys

LIST Keys

Using a key mapping directive allows you to assign a different or additional key to perform the function described. For example, to use function key **F3** to invoke the HELP facility (normally invoked with **F1**):

```
Help = F3
```

Any directive can be used multiple times to assign multiple keys to the same function. For example:

```
ListFind = F            ;F does a find in LIST
ListFind = F5           ;F5 also does a find in LIST
```

Use some care when you reassign keystrokes. If you assign a default key to a different function, it will no longer be available for its original use. For example, if you assign **F1** to the AddFile directive (a part of filename completion), the **F1** key will no longer invoke the help system, so you will probably want to assign a different key to Help.

See Keys and Key Names before using the key mapping directives.

Key assignments are processed before looking for keystroke aliases. For example, if you assign **Shift-F1** to HELP and also assign **Shift-F1** to a key alias, the key alias will be ignored.

Assigning a new keystroke for a function does not deassign the default keystroke for the same function. If you want to deassign one of the default keys, use the NormalKey directive described below or the corresponding directive for keys in the other key groups (NormalEditKey, NormalHWinKey, or NormalListKey).

General Input Keys

These directives apply to all input.   They are in effect whenever 4NT requests input from the keyboard, including during command-line editing and the DESCRIBE, ESET, INPUT, LIST, and SELECT commands.   The general input keys are:

Backspace	Deletes the character to the left of the cursor
BeginLine	Moves the cursor to the start of the line
Del	Deletes the character at the cursor
DelToBeginning	Deletes from the cursor to the start of the line
DelToEnd	Deletes from the cursor to the end of the line
DelWordLeft	Deletes the word to the left of the cursor
DelWordRight	Deletes the word to the right of the cursor
Down	Moves the cursor or scrolls the display down
EndLine	Moves the cursor to the end of the line
EraseLine	Deletes the entire line
ExecLine	Executes or accepts a line
Ins	Toggles insert / overstrike mode
Left	Moves the cursor or scrolls the display left
NormalKey	Deassigns a key
Right	Moves the cursor or scrolls the display right
Up	Moves the cursor or scrolls the display up
WordLeft	Moves the cursor left one word
WordRight	Moves the cursor right one word

Backspace = Key (Bksp): Deletes the character to the left of the cursor.

BeginLine = Key (Home): Moves the cursor to the beginning of the line.

Del = Key (Del): Deletes the character at the cursor.

DelToBeginning = Key (Ctrl-Home):   Deletes from the cursor to the start of the line.

DelToEnd = Key (Ctrl-End): Deletes from the cursor to the end of the line.

DelWordLeft = Key (Ctrl-L): Deletes the word to the left of the cursor.

DelWordRight = Key (Ctrl-R, Ctrl-Bksp): Deletes the word to the right of the cursor. See ClearKeyMap if you need to remove the default mapping of **Ctrl-Bksp** to this function.

Down = Key (Down): Scrolls the display down one line in LIST; moves the cursor down one line in SELECT and in the command-line history, directory history, or %@SELECT window. (Scrolling down through the command history at the prompt is controlled by NextHistory, not by this directive.)

EndLine = Key (End): Moves the cursor to the end of the line.

EraseLine = Key (Esc):   Deletes the entire line.

ExecLine = Key (Enter): Executes or accepts a line.

Ins = Key (Ins): Toggles insert / overstrike mode during line editing.

Left = Key (Left): Moves the cursor left one character on the input line; scrolls the display left 8 columns in LIST; scrolls the display left 4 columns in the command-line, directory history, or %@SELECT window.

NormalKey = Key: Deassigns a general input key in order to disable the usual meaning of the key within 4NT and/or make it available for keystroke aliases. This will make the keystroke operate as a "normal" key with no special function. For example:

```
NormalKey = Ctrl-End
```

will disable Ctrl-End, which is the standard "delete to end of line" key. Ctrl-End could then be assigned to a keystroke alias. Another key could be assigned the "delete to end of line" function with the DelToEnd directive.

Right = Key (Right): Moves the cursor right one character on the input line; scrolls the display right 8 columns in LIST; scrolls the display right 4 columns in the command-line history, directory history, or %@SELECT window.

Up = Key (Up): Scrolls the display up one line in LIST; moves the cursor up one line in SELECT and in the command-line history, directory history, or %@SELECT window. (Scrolling up through the command history at the prompt is controlled by PrevHistory, not by this directive.)

WordLeft = Key (Ctrl-Left):   Moves the cursor left one word; scrolls the display left 40 columns in LIST.

WordRight = Key (Ctrl-Right):   Moves the cursor right one word; scrolls the display right 40 columns in LIST.

Command-Line Editing Keys

These directives apply only to command-line editing.   They are only effective at the 4NT prompt.   The command-line editing keys are:

AddFile	Keeps filename completion entry and adds another
AliasExpand	Expands aliases without executing them
CommandEscape	Allows direct entry of a keystroke
DelHistory	Deletes a history list entry
EndHistory	Displays the last entry in the history list
Help	Invokes this help system
LFNToggle	Switches filename completion between LFN and SFN modes
LineToEnd	Copies command line to end of history, and executes it
NextFile	Gets the next matching filename
NextHistory	Recalls the next command from the history
NormalEditKey	Deassigns a command-line editing key
PopFile	Opens the filename completion window
PrevFile	Gets the previous matching filename
PrevHistory	Recalls the previous command from the history
SaveHistory	Saves the command line without executing it

AddFile = Key (F10): Keeps the current filename completion entry and inserts the next matching name.

AliasExpand = Key (Ctrl-F):   Expands all aliases in the current command line without executing them.

CommandEscape = Key (Alt-255): Allows direct entry of a keystroke that would normally be handled by the command line editor (e.g. **Tab** or **Ctrl-D**).

DelHistory = Key (Ctrl-D): Deletes the displayed history list entry and displays the previous entry.

EndHistory = Key (Ctrl-E):   Displays the last entry in the history list.

Help = Key (F1): Invokes the HELP facility.

LFNToggle = Key (Ctrl-A): Toggles filename completion between long filename and short filename modes on LFN drives.

LineToEnd = Key (Ctrl-Enter): Copies the current command line to the end of the history list, then executes it.

NextFile = Key (F9, Tab):   Gets the next matching filename. See ClearKeyMap if you need to remove the default mapping of **Tab** to this function.

NextHistory = Key (Down): Recalls the next command from the command history.

NormalEditKey = Key: Deassigns a command-line editing key in order to disable the usual meaning of the key while editing a command line, and/or make it available for keystroke aliases. For additional details see NormalKey.

PopFile = Key (F7, Ctrl-Tab): Opens the filename completion window. You may not be able to use **Ctrl-Tab**, because not all systems recognize it as a keystroke. See ClearKeyMap if you need to remove the default mapping of **Ctrl-Tab** to this function.

PrevFile = Key (F8, Shift-Tab): Gets the previous matching filename. See ClearKeyMap if you need to remove the default mapping of **Shift-Tab** to this function.

PrevHistory = Key (Up): Recalls the previous command from the command history.

SaveHistory = Key (Ctrl-K): Saves the command line in the command history list without executing it.

Popup Window Keys

These directives apply to popup windows, including the command history window, the directory history window, the filename completion window, the extended directory search window, and the @SELECT window.   The popup window keys are:

DirWinOpen	Opens the directory history window
HistWinOpen	Opens the command history window
NormalPopupKey	Deassigns a popup window key
PopupWinBegin	Moves to the first line of the popup window
PopupWinDel	Deletes a line from within the popup window
PopupWinEdit	Moves a line from the popup window to the prompt
PopupWinEnd	Moves to the last line of the popup window
PopupWinExec	Executes the selected line in the popup window

DirWinOpen = Key (Ctrl-PgUp): Opens the directory history window while at the command line.

HistWinOpen = Key (PgUp): Brings up the history window while at the command line.

NormalPopupKey = Key: Deassigns a popup window key in order to disable the usual meaning of the key within the popup window. For additional details see the NormalKey directive.

PopupWinBegin = Key (Ctrl-PgUp): Moves to the first item in the list when in the popup window.

PopupWinDel = Key (Ctrl-D): Deletes a line from within the command history or directory history window.

PopupWinEdit = Key (Ctrl-Enter): Moves a line from the command history or directory history window to the prompt for editing.

PopupWinEnd = Key (Ctrl-PgDn): Moves to the last item in the list when in the popup window.

PopupWinExec = Key (Enter): Selects the current item and closes the window.

LIST Keys

These directives are effective only inside the LIST command.   The LIST keys are:

ListExit	Exits the current file
ListFind	Prompts and searches for a string
ListFindReverse	Prompts and searches backward for a string
ListHex	Toggles hexadecimal display mode
ListHighBit	Toggles LIST's "strip high bit" option
ListInfo	Displays information about the current file
ListNext	Finds the next matching string
ListPrevious	Finds the previous matching string
ListPrint	Prints the file on LPT1
ListWrap	Toggles LIST's wrap option
NormalListKey	Deassigns a LIST key

ListExit = Key (Esc): Exits from the LIST command.

ListFind = Key (F):   Prompts and searches for a string.

ListFindReverse = Key (Ctrl-F): Prompts and searches backward for a string.

ListHex = Key (X):   Toggles hexadecimal display mode.

ListHighBit = Key (H):   Toggles LIST's "strip high bit" option, which can aid in displaying files from certain word processors.

ListInfo = Key (I): Displays information about the current file.

ListNext = Key (N): Finds the next matching string.

ListPrevious = Key (Ctrl-N): Finds the previous matching string.

ListPrint = Key (P):   Prints the file on LPT1.

ListWrap = Key (W):   Toggles LIST's wrap option on and off. The wrap option wraps text at the right margin.

NormalListKey = Key: Deassigns a LIST key in order to disable the usual meaning of the key within LIST. For additional details see NormalKey.

Advanced Directives

These directives are generally used for unusual circumstances, or for diagnosing problems. Most often they are not needed in normal use. The advanced directives are:

ClearKeyMap	Clear default key mappings
Debug	Set debugging options
Include	Include a file containing *.INI* directives
NextINIFile	Set secondary shell *.INI* file name

ClearKeyMap: Clears all current key mappings. ClearKeyMap is a special directive which has no value or "=" after it. Use ClearKeyMap to make one of the keys in the default map (**Tab**, **Shift-Tab**, **Ctrl-Tab**, or **Ctrl-Bksp**) available for a keystroke alias, or in the **[Secondary]** section of the *.INI* file to clear key mappings inherited from the primary shell. ClearKeyMap should appear before any key mapping directives. If you want to clear some but not all of the default mappings, use ClearKeyMap, then recreate the mappings you want to retain (*e.g.*, with "NextFile=Tab", etc.).

Debug = nnnn (0): Controls certain debugging options which can assist you in tracking down unusual problems. Use the following values for Debug; to select more than one option, add the values together:

1	During the startup process, display the complete command tail passed to 4NT, then wait for a keystroke.
2	Include the product name with each error message displayed by 4NT. This may be useful if you are unsure of the origin of a particular error message.

Also see the batch file debugger, a separate and unrelated facility for stepping through batch files.

Include = File: Include the text from the named file at this point in the processing of the current *.INI* file. Use this option to share a file of directives between several products. The text in the named file is processed just as if it were part of the original *.INI* file. When the include file is finished, processing resumes at the point where it left off in the original file. The included file may contain any valid directive for the current section, but may not contain a section name. Includes may be nested up to three levels deep (counting the original file as level 1).

You must maintain include files manually — the OPTION command modifies the original *.INI* file only, and does not update included files.

NextINIFile = File: The full path and name of the file must be specified. All subsequent shells will read the specified *.INI* file, and ignore any **[Secondary]** section in the original *.INI* file.

4NT Commands by Name

?
ACTIVATE
ALIAS
ATTRIB
ASSOC
BEEP
CALL
CANCEL
CD / CHDIR
CDD
CLS
COLOR
COPY
DATE
DEL / ERASE
DELAY
DESCRIBE
DETACH
DIR
DIRHISTORY
DIRS
DO
DRAWBOX
DRAWHLINE
DRAWVLINE
ECHO
ECHOERR
ECHOS
ECHOSERR
ENDLOCAL
ESET
EXCEPT
EXIT
FFIND
FOR
FREE
FTYPE
GLOBAL
GOSUB
GOTO
HELP
HISTORY
IF
IFF
INKEY
INPUT
KEYBD
KEYS
LIST
LOADBTM
LOG
MD / MKDIR
MEMORY
MOVE
MSGBOX
ON
OPTION
PATH
PAUSE
POPD
PROMPT
PUSHD
QUIT
RD / RMDIR
REBOOT
REM
REN / RENAME
RETURN
SCREEN
SCRPUT
SELECT
SET
SETDOS
SETLOCAL
SHIFT
SHRALIAS
START
SWITCH
TEE
TEXT
TIME
TIMER
TITLE
TOUCH
TREE
TRUENAME
TYPE
UNALIAS
UNSET
VER
VERIFY
VOL
VSCRPUT
WINDOW
Y

4NT Commands

The best way to learn the 4NT commands is to experiment with them.   The lists below categorize the available commands by topic and will help you find the ones that you need.

System configuration

CLS
COLOR
DATE
DIRHISTORY
FREE
HISTORY
KEYBD
KEYS
LOG
MEMORY
OPTION
PROMPT
REBOOT
SETDOS
TIME
VER
VERIFY
VOL

File and directory management

ATTRIB
COPY
DEL / ERASE
DESCRIBE
FFIND
LIST
MOVE
REN / RENAME
SELECT
TOUCH
TREE
TRUENAME
TYPE

Subdirectory management

CD / CHDIR
CDD
DIR
DIRS
MD / MKDIR
POPD
PUSHD
RD / RMDIR

Input and output

DRAWBOX	ECHOERR	INPUT	VSCRPUT
DRAWHLINE	ECHOS	MSGBOX	
DRAWVLINE	ECHOSERR	SCREEN	
ECHO	INKEY	SCRPUT	

Commands primarily for use in or with batch files and aliases (some work only in batch files; see the individual commands for details)

ALIAS	ENDLOCAL	IFF	RETURN
BEEP	FOR	LOADBTM	SETLOCAL
CALL	GLOBAL	ON	SHIFT
CANCEL	GOSUB	PAUSE	SWITCH
DELAY	GOTO	QUIT	TEXT
DO	IF	REM	UNALIAS

Environment and path commands

ESET
PATH
SET
UNSET

Other commands

?	EXCEPT	SHRALIAS	TITLE
ACTIVATE	EXIT	START	WINDOW
ASSOC	FTYPE	TEE	Y
DETACH	HELP	TIMER	

?

Purpose: Display a list of internal commands, or prompt for a command.

Format: **? ["prompt" command]**

prompt: Prompt text about whether to execute the **command**.
command: Command to be executed if user answers **Y**.

Usage

? has two functions. When you use the **?** by itself, it displays a list of internal commands. If you have disabled a command with SETDOS /I, it will not appear in the list.

The second function of **?** is to prompt the user before executing a specific line in a batch file. If you add a **prompt** and a **command**, **?** will display the prompt followed by "(Y/N)?" and wait for the user's response. If the user presses "Y" or "y", the line will be executed. If the user presses "N" or "n", the line will be ignored.

For example, the following command might be used in a batch file:

```
? Load the network call netstart.btm
```

When this command is executed, you will see the following prompt; if you answer "Y", the CALL command will be executed:

```
Load the network (Y/N)?
```

ACTIVATE

Purpose: Activate a window, set its state, or change its title.

Format: **ACTIVATE "window" [MAX | MIN | RESTORE | CLOSE | "title"]**

window: Current title of window to work with.
title: New title for window.

See also: START, TITLE, and WINDOW.

Usage

Both the current name of the window and the new name, if any, must be enclosed in double quotes. The quotes will not appear as part of the title bar text.

If no options are used, the window named in the command will become the active window and be able to receive keystrokes and mouse commands.

The MAX option expands the window to its maximum size, the MIN option reduces the window to an icon, and the RESTORE option returns the window to its default size and location on the desktop. The CLOSE option closes the window and ends the session running in the window.

This example maximizes and then renames the window called "4NT":

```
[c:\] activate "4NT" max
[c:\] activate "4NT" "Command Prompt"
```

You can use wildcards in the ***window*** name if you only know the first part of the title. This is useful with applications that change their window title to reflect the file currently in use.

ALIAS

Purpose: Create new command names that execute one or more commands or redefine default options for existing commands; assign commands to keystrokes; load or display the list of defined alias names.

Format: **ALIAS [/P /R *file*...] [*name* [=][*value*]]**

file: One or more files to read for alias definitions.
name: Name for an alias, or for the key to execute the alias.
value: Text to be substituted for the alias name.

/P(ause) **/R**(ead file)

See also: UNALIAS and Aliases.

Usage

The ALIAS command lets you create new command names or redefine internal commands. It also lets you assign one or more commands to a single keystroke. An alias is often used to execute a complex series of commands with a few keystrokes or to create "in memory batch files" that run much faster than disk-based batch files.

For example, to create a single-letter command D to display a wide directory, instead of using the longer DIR /W, you could use the command:

```
[c:\] alias d = dir /w
```

Now when you type a single **d** as a command, it will be translated into a DIR /W command.

If you define aliases for commonly used application programs, you can often remove the directories they're stored in from the PATH. For example, if you use Quattro Pro and had the *C:\QPRO* directory in your path, you could define the following alias:

```
[c:\] alias qpro = c:\qpro\q.exe
```

With this alias defined, you can probably remove *C:\QPRO* from your PATH. Quattro Pro will now load more quickly than it would if 4NT had to search the PATH for it. In addition, the PATH can be shorter, which will speed up searches for other programs.

If you apply this technique for each application program, you can often reduce your PATH to just two or three directories containing utility programs, and significantly reduce the time it takes to load most software on your system. Before removing a directory from the PATH, you will need to define aliases for all the executable programs you commonly use which are stored in that directory.

Aliases are stored in memory, and are not saved automatically when you turn off your computer or end your current session. See below for information on saving and reloading your aliases.

Multiple Commands and Special Characters in Aliases

An alias can represent more than one command. For example:

```
[c:\] alias letters = `cd \letters & text`
```

creates a new command called LETTERS. The command first uses CD to change to a subdirectory called *\LETTERS* and then runs a program called *TEXT*. The ampersand [**&**] is the command separator and indicates that the two commands are distinct and should be executed sequentially.

Aliases make extensive use of the command separator, and the parameter character, and may also use the escape character. These characters differ between 4DOS and 4OS2 or 4NT. In the text and examples below, we use the 4NT characters. If you want to use the same aliases under different command processors, see Special Character Compatibility.

When you type alias commands at the command line or in a batch file, you **must** use back quotes [`] around the definition if it contains multiple commands, parameters (discussed below), environment variables, redirection, or piping. The back quotes prevent premature expansion of these arguments. You **may** use back quotes around other definitions, but they are not required. (You do not need back quotes when your aliases are loaded from an ALIAS /R file; see below for details.) The examples above and below include back quotes only when they are required.

Nested Aliases

Aliases may invoke internal commands, external commands, or other aliases. (However, an alias may not invoke itself, except in special cases where an IF or IFF command is used to prevent an infinite loop.) The two aliases below demonstrate alias nesting (one alias invoking another). The first line defines an alias which runs a program called *WP.EXE* that is in the *E:\WP60\* subdirectory. The second alias changes directories with the PUSHD command, runs the WP alias, and then returns to the original directory with the POPD command:

```
[c:\] alias wp = e:\wp60\wp.exe
[c:\] alias w = `pushd c:\wp & wp & popd`
```

The second alias above could have included the full path and name of the *WP.EXE* program instead of calling the WP alias. However, writing two aliases makes the second one easier to read and understand, and makes the first alias available for independent use. If you rename the *WP.EXE* program or move it to a new directory, only the first alias needs to be changed.

Temporarily Disabling Aliases

If you put an asterisk [*****] immediately before a command in the *value* of an alias definition (the part after the equal sign), it tells 4NT not to attempt to interpret that command as another (nested) alias. An asterisk used this way must be preceded by a space or the command separator and followed immediately by an internal or external command name.

By using an asterisk, you can redefine the default options for any internal or external command. For example, suppose that you always want to use the DIR command with the **/2** (two column) and **/P** (pause at the end of each page) options. The following line will do just that:

```
[c:\] alias dir = *dir /2/p
```

If you didn't include the asterisk, the second DIR on the line would be the name of the alias itself, and 4NT would repeatedly re- invoke the DIR alias, rather than running the DIR command. This would cause an "Alias loop" or "Command line too long" error.

An asterisk also helps you keep the names of internal commands from conflicting with the names of external programs. For example, suppose you have a program called *LIST.COM*. Normally, the internal LIST command will run anytime you type LIST. But two simple aliases will give you access to both the *LIST.COM* program and the LIST command:

```
[c:\] alias list = c:\util\list.com
[c:\] alias display = *list
```

The first line above defines LIST as an alias for the *LIST.COM* program. If you stopped there, the external program would run every time you typed LIST and you would not have easy access to the internal LIST command. The second line renames the internal LIST command as DISPLAY. The asterisk is needed in the second command to indicate that the following word means the internal command LIST, not the LIST alias which runs your external program.

Another way to understand the asterisk is to remember that a command is always checked for an alias first, then for an internal or external command, or a batch file. The asterisk at the beginning of a command name simply skips over the usual check for aliases when processing that command, and allows the command processor to go straight to checking for an internal command, external command, or batch file.

You can also use an asterisk before a command that you enter at the command line or in a batch file. If you do, that command won't be interpreted as an alias. This can be useful when you want to be sure you are running the true, original command and not an alias with the same name, or temporarily defeat the purpose of an alias which changes the meaning or behavior of a command.

You can also disable aliases temporarily with the SETDOS /X command.

Partial Alias Names

You can also use an asterisk in the *name* of an alias. When you do, the characters following the asterisk are optional when you invoke the alias command. (Use of an asterisk in the alias *name* is unrelated to the use of an asterisk in the alias *value* discussed above.) For example, with this alias:

```
[c:\] alias wher*eis = dir /sp
```

the new command, WHEREIS, can be invoked as WHER, WHERE, WHEREI, or WHEREIS. Now if you type:

```
[c:\] where myfile.txt
```

The WHEREIS alias will be expanded to the command:

```
dir /sp myfile.txt
```

Keystroke Aliases

If you want to assign an alias to a keystroke, use the keyname on the left side of the equal sign, preceded by an at sign [**@**]. For example, to assign the command DIR /W to the **F5** key, type

```
[c:\] alias @F5 = dir /w
```

See Keys and Key Names for a complete listing of key names and a description of the key name format.

When you define keystroke aliases, the assignments will only be in effect at the command line, not inside application programs. Be careful not to assign aliases to keys that are already used at the command line (like **F1** for Help). The command-line meanings take precedence and the keystroke alias will never be invoked. If you want to use one of the command-line keys for an alias instead of its normal meaning, you must first disable its regular use with the NormalKey or NormalEditKey directives in your *.INI* file.

If you define a keystroke alias with a single at sign as shown above, then, when you press the **F5** key, the value of the alias (DIR /W above) will be placed on the command line for you. You can type additional parameters if you wish and then press **Enter** to execute the command. With this particular alias, you can define the files that you want to display after pressing **F5** and before pressing Enter to execute the command.

If you want the keystroke alias to take action automatically without waiting for you to edit the command line or press **Enter**, you can begin the definition with two at signs [**@@**]. 4NT will execute the alias "silently," without displaying its text on the command line. For example, this command will assign an alias to the **F6** key that uses the CDD command to take you back to the previous default directory:

```
        [c:\] alias @@f6 = cdd -
```

You can also define a keystroke alias by using "@" or "@@" plus a scan code for one of the permissible keys (see the Key Code Tables for a list of scan codes). In most cases it will be easier to use key names. Scan codes should only be used with unusual keyboards where a key name is not available for the key you are using.

Displaying Aliases

If you want to see a list of all current ALIAS commands, type:

```
        [c:\] alias
```

You can also view the definition of a single alias. If you want to see the definition of the alias LIST, you can type:

```
        [c:\] alias list
```

Saving and Reloading Your Aliases

You can save your aliases to a file called *ALIAS.LST* this way:

```
        [c:\] alias > alias.lst
```

You can then reload all the alias definitions in the file the next time you boot up with the command:

```
        [c:\] alias /r alias.lst
```

This is much faster than defining each alias individually in a batch file. If you keep your alias definitions in a separate file which you load when your system starts, you can edit them with a text editor, reload the edited file with ALIAS /R, and know that the same alias list will be loaded the next time you boot your computer.

When you define aliases in a file that will be read with the ALIAS /R command, you do not need back quotes around the value, even if back quotes would normally be required when defining the same alias at the command line or in a batch file.

To remove an alias, use the UNALIAS command.

Alias Parameters

Aliases can use command-line arguments or parameters like those in batch files. The command-line arguments are numbered from **%0** to **%127**. **%0** contains the alias name. It is up to the alias to determine the meaning of the other parameters. You can use quotation marks to pass spaces, tabs, commas, and other special characters in an alias parameter; see Argument Quoting for details.

Parameters that are referred to in an alias, but which are missing on the command line, appear as empty strings inside the alias. For example, if you put two parameters on the command line, any reference in the alias to **%3** or any higher-numbered parameter will be interpreted as an empty string.

The parameter **%n$** has a special meaning. 4NT interprets it to mean "the entire command line, from argument **n** to the end." If **n** is not specified, it has a default value of 1, so **%$** means "the entire command line after the alias name." The special parameter **%#** contains the number of command-line arguments.

For example, the following alias will change directories, perform a command, and return to the original directory:

```
[c:\] alias in `pushd %1 & %2$ & popd`
```

When this alias is invoked as:

```
[c:\] in c:\comm mycomm /zmodem /56K
```

the first parameter, **%1**, has the value *c:\comm*. **%2** is *mycomm*, **3**

is */zmodem*, and **%4** is */56K*. The command line expands into these three separate commands:

```
pushd c:\comm
ycomm /zmodem /56K
popd
```

This next example uses the IFF command to redefine the defaults for SET. It should be entered on one line:

```
[c:\] alias set = `iff %# == 0 then & *set /p & else & *set %& & endiff`
```

This modifies the SET command so that if SET is entered with no arguments, it is replaced by SET /P (pause after displaying each page), but if SET is followed by an argument, it behaves normally. Note the use of asterisks (***set**) to prevent alias loops.

If an alias uses parameters, command-line arguments will be deleted up to and including the highest referenced argument. For example, if an alias refers only to **%1** and **%4**, then the first and fourth arguments will be used, the second and third arguments will be discarded, and any additional arguments beyond the fourth will be appended to the expanded command (after the *value* portion of the alias). If an alias uses no parameters, all of the command- line arguments will be appended to the expanded command.

Aliases also have full access to all variables in the environment, internal variables, and variable functions. For example, you can create a simple command-line calculator this way (enter this on one line):

```
[c:\] alias calc = `echo The answer is: %@eval[%&]`
```

Now, if you enter:

```
[c:\] calc 5 * 6
```

the alias will display:

```
The answer is: 30
```

Expanding Aliases

You can expand an alias on the command line and view or edit the results by pressing **Ctrl-F** after typing the alias name, but before the command is executed. This replaces the alias with its contents, and substitutes values for each alias paramter, just as if you had pressed the **Enter** key. However, the command is not executed; it is simply redisplayed on the command line for additional editing.

Ctrl-F is especially useful when you are developing and debugging a complex alias, or if you want to make sure that an alias that you may have forgotten won't change the effect of your command.

Local and Global Aliases

The aliases can be stored in either a "local" or "global" list.

With a local alias list, any changes made to the aliases will only affect the current copy of 4NT. They will not be visible in other shells or other sessions.

With a global alias list, all copies of 4NT will share the same alias list, and any changes made to the aliases in one copy will affect all other copies. This is the default.

You can control the type of alias list with the LocalAliases directive in the *.INI* file, and with the **/L** and **/LA** options of the START command.

Whenever you start a secondary shell which uses a local alias list, it inherits a copy of the aliases from the previous shell. However, any changes to the aliases made in the secondary shell will affect only that shell. If you want changes made in a secondary shell to affect the previous shell, use a global alias list in both shells.

Retaining Global Aliases with SHRALIAS

If you select a global alias list for 4NT you can share the aliases among all copies of 4NT running in any session. When you close all 4NT sessions, the memory for the global alias list is released, and a new, empty alias list is created the next time you start 4NT.

If you want the alias list to be retained in memory even when no command processor session is running, execute the SHRALIAS command, which loads a program to perform this service for the global alias list, the global command history list, and the global directory history.

SHRALIAS retains the alias list in memory, but cannot preserve it when Windows NT itself is shut down. To save your aliases when restarting NT, you must store them in a file and reload them after the system restarts. For details on how to do so, see **Saving and Reloading Your Aliases** (above).

The UNKNOWN_CMD Alias

If you create an alias with the name **UNKNOWN_CMD**, it will be executed any time 4NT would normally issue an "Unknown command" error message. This allows you to define your own handler for unknown commands. When the **UNKNOWN_CMD** alias is executed, the command line which generated the error is passed to the alias for possible processing. For example, to display the command that caused the error:

```
     alias unknown_cmd `echo Error in command "%&"`
```

If the **UNKNOWN_CMD** alias contains an unknown command, it will call itself repeatedly. If this occurs, the command processor will loop up to 10 times, then display an "UNKNOWN_CMD loop" error.

Options

/P (Pause) This option is only effective when ALIAS is used to display existing definitions. It pauses the display after each page and waits for a keystroke before continuing (see Page and File Prompts).

/R (Read file) This option loads an alias list from a file. The format of the file is the same as that of the ALIAS display:

```
   name=value
```

where **name** is the *name* of the alias and **value** is its *value*.   You can use an equal sign [**=**] or space to separate the name and value.   Back quotes are not required around the value. You can add comments to the file by starting each comment line with a colon [**:**].   You can load multiple files with one ALIAS /R command by placing the names on the command line, separated by spaces:

```
   [c:\] alias /r alias1.lst alias2.lst
```

Each definition in an ALIAS /R file can be up to 2047 characters long. The definitions can span multiple lines in the file if each line, except the last, is terminated with an escape character.

ASSOC

Purpose: Modify or display relationships between file extensions and file types stored in the Windows NT registry.

Format: **ASSOC [/P] [*.ext*[=[*filetype*]]]**

.ext: The file extension whose file type you want to display or set.
filetype: A file type stored in the Windows NT registry.

/P(ause)

See also: FTYPE and Executable Extensions.

Usage

ASSOC allows you to create, modify, or display associations between file extensions and file types stored in the Windows NT registry.

ASSOC manages "indirect" Windows NT file associations stored under the registry handle HKEY_CLASSES_ROOT, and discussed in more detail under Windows File Associations and Using Windows File Associations. If you are not familiar with file associations be sure to read about them before using ASSOC.

If you invoke ASSOC with no parameters, it will display the current associations. If you include a ***.ext***, with no equal sign or ***filetype***, ASSOC will display the current association for that extension.

If you include the equal sign and ***filetype***, ASSOC will create or update the association for extension ***.ext*** to refer to the specified file type. The valid filetypes depend on the contents of your Windows NT registry. See the FTYPE command or your Windows NT documentation for additional details.

ASSOC cannot delete an extension from the registry. However, you can create a similar effect by associating the extension with an empty file type using **ASSOC .ext=**, without the ***filetype*** parameter.

ASSOC should be used with caution, and only after backing up the registry. Improper changes to file associations can prevent applications and / or the operating system from working properly.

Options

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

ATTRIB

Purpose: Change or view file and subdirectory attributes.

Format: **ATTRIB [/A:[[-]rhsda] /D /E /P /Q /S] [+|-[AHRS]]** ***file*** **...**

file: A file, directory, or list of files or directories on which to operate.

/A: (Attribute select)	**/P**(ause)
/D(irectories)	**/Q**(uiet)
/E (No error messages)	**/S**(ubdirectories)

Attribute flags:

+A	Set the archive attribute
-A	Clear the archive attribute
+H	Set the hidden attribute
-H	Clear the hidden attribute
+R	Set the read-only attribute
-R	Clear the read-only attribute
+S	Set the system attribute
-S	Clear the system attribute

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

Every file and subdirectory has 4 attributes that can be turned on (set) or turned off (cleared): **Archive**, **Hidden**, **Read-only**, and **System**.

The ATTRIB command lets you view, set, or clear attributes for any file, group of files, or subdirectory. You can view file attributes by entering ATTRIB without specifying new attributes (*i.e.*, without the **[+|-[AHRS]]** part of the format). (You can also view file attributes with the **DIR /T** command).

The primary use of ATTRIB is to set attributes. For example, you can set the read-only and hidden attributes for the file *MEMO*:

```
[c:\] attrib +rh memo
```

Attribute options apply to the file(s) that follow the options on the ATTRIB command line. The example below shows how to set different attributes on different files with a single command. It sets the archive attribute for all *.TXT* files, then sets the system attribute and clears the archive attribute for *TEST.COM*:

```
[c:\] attrib +a *.txt +s -a test.com
```

When you use ATTRIB on an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names.

To change directory attributes, use the **/D** switch. If you give ATTRIB a directory name instead of a file name, and omit **/D**, it will append "*.*" to the end of the name and act on all files in that directory, rather than acting on the directory itself.

Your operating system also supports "D" (subdirectory) and "V" (volume label) attributes. These attributes cannot be altered with ATTRIB; they are designed to be controlled only by the operating system itself.

ATTRIB will ignore underlines in the new attribute (the **[+|-[AHRS]]** part of the command). For example, ATTRIB sees these 2 commands as identical:

```
[c:\] attrib +a filename
[c:\] attrib +__A_ filename
```

This allows you to use a string of attributes from either the @ATTRIB variable function or from ATTRIB itself (both of which use underscores to represent attributes that are not set) and send that string back to ATTRIB to set attributes for other files. For example, to clear the attributes of *FILE2* and then set its attributes to match those of *FILE1* (enter this on one line):

```
[c:\] attrib -arhs file2 & attrib +%@attrib[file1] file2
```

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **ATTRIB /A: ...**), ATTRIB will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

The **/A:** switch specifies which files to select, **not** which attributes to set. For example, to remove the archive attribute from all hidden files, you could use this command:

```
[c:\] attrib /a:h -a *.*
```

/D: (Directories) If you use the **/D** option, ATTRIB will modify the attributes of subdirectories in addition to files (yes, you can have a hidden subdirectory):

```
[c:\] attrib /d +h c:\mydir
```

If you use a directory name instead of a file name, and omit **/D**, ATTRIB will append "*.*" to the end of the name and act on all files in that directory, rather than acting on the directory itself.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) This option turns off ATTRIB's normal screen output. It is most useful in batch files.

/S (Subdirectories) If you use the **/S** option, the ATTRIB command will be applied to all matching files in the current or named directory and all of its subdirectories.

BEEP

Purpose: Beep the speaker or play simple music.

Format: **BEEP [*frequency duration* ...]**

frequency: The beep frequency in Hertz (cycles per second).
duration: The beep length in 1/18th second intervals.

Usage

BEEP generates a sound through your computer's speaker. It is normally used in batch files to signal that an operation has been completed, or that the computer needs attention.

Because BEEP allows you to specify the frequency and duration of the sound, you can also use it to play simple music or to create different kinds of signals for the user.

You can include as many frequency and duration pairs as you wish. No sound will be generated for frequencies less than 20 Hz, allowing you to use BEEP as a way to create short delays. The default value for *frequency* is 440 Hz; the default value for *duration* is 2.

This batch file fragment runs a program called *DEMO*, then plays a few notes and waits for you to press a key:

```
demo & beep 440 4  600 2  1040 6
pause Finished with the demo - hit a key...
```

The following table gives the *frequency* values for a five octave range (middle C is 262 Hz):

C	131	262	523	1046	2093
C# / Db	139	277	554	1108	2217
D	147	294	587	1175	2349
D# / Eb	156	311	622	1244	2489
E	165	330	659	1318	2637
F	175	349	698	1397	2794
F# / Gb	185	370	740	1480	2960
G	196	392	784	1568	3136
G# / Ab	208	415	831	1662	3322
A	220	440	880	1760	3520
A# / Bb	233	466	932	1866	3729
B	248	494	988	1973	3951

CALL

Purpose: Execute one batch file from within another.

Format: **CALL *file***

file: The batch file to execute.

See also: CANCEL and QUIT.

Usage

CALL allows batch files to call other batch files (batch file nesting).   The calling batch file is suspended while the called (second) batch file runs.   When the second batch file finishes, the original batch file resumes execution at the next command.   If you execute a batch file from inside another batch file without using CALL, the first batch file is terminated before the second one starts.

4NT supports batch file nesting up to ten levels deep.

The current ECHO state is inherited by a called batch file.

The called batch file should always either return (by executing its last line, or using the QUIT command), or terminate batch file processing with CANCEL.   Do not restart or CALL the original batch file from within the called file as this may cause an infinite loop or a stack overflow.

CALL returns an exit code which matches the batch file return code. You can test this exit code with the %_? or %? environment variable, and use it with conditional commands.

CANCEL

Purpose: Terminate batch file processing.

Format: **CANCEL [*value*]**

value: The numeric exit code to return to 4NT.

See also: CALL and QUIT.

Usage

The CANCEL command ends all batch file processing, regardless of the batch file nesting level. Use QUIT to end a nested batch file and return to the previous batch file.

You can CANCEL at any point in a batch file. If CANCEL is used from within an alias it will end execution of both the alias and any batch files which are running at the time.

The following batch file fragment compares an input line to "end" and terminates all batch file processing if it matches:

```
input Enter your choice:  %%option
if "%option" == "end" cancel
```

If you specify a *value*, CANCEL will set the ERRORLEVEL or exit code to that value (see the IF command, and the %? variable).

CD / CHDIR

Purpose: Display or change the current directory.

Format: **CD [/D] [*path* | -]**

or

CHDIR [/D] [*path* | -]

path: The directory to change to, including an optional drive name.

/D(rive)

See also: CDD, MD, PUSHD, RD, CDPATH, and Directory Navigation.

Usage

CD and CHDIR are synonyms. You can use either one.

CD lets you navigate a drive's subdirectory structure by changing the current working directory. If you enter CD and a directory name, the named directory becomes the new current directory. For example, to change to the subdirectory *C:\FINANCE\MYFILES*:

```
[c:\] cd \finance\myfiles
[c:\finance\myfiles]
```

Every disk drive on the system has its own current directory. Specifying both a drive and a directory in the CD command will change the current directory on the specified drive, but will not change the default drive. For example, to change the default directory on drive A:

```
[c:\] cd a:\utility
[c:\]
```

Notice that this command does not change to drive A:. Use the CDD command to change the current drive and directory at the same time.

When you use CD to change to a directory on an HPFS, NTFS, or LFN drive, you must quote the *path* name if it contains whitespace or special characters. See File Names for additional details.

You can change to the parent directory with **CD ..**; you can also go up one additional directory level with each additional [**.**]. For example, **CD ....** will go up three levels in the directory tree (see Extended Parent Directory Names). You can move to a sibling directory -- one that branches from the same parent directory as the current subdirectory -- with a command like **CD .\newdir .**

If you enter CD with no argument or with only a disk drive name, it will display the current directory on the default or named drive.

If CD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database in order to find a matching directory and switch to it. You can also use wildcards in the *path* to force an extended directory search. See Directory Navigation for complete details on these and other directory navigation features.

CD saves the current directory before changing to a new directory. You can switch back to the previous directory by entering **CD -** (there must be a space between the CD command and the hyphen). You can switch back and forth between two directories by repeatedly entering **CD -**. The saved directory is the same for both the CD and CDD commands. Drive changes and automatic directory changes also modify the saved directory, so you can use **CD -** to return to a directory that you exited with an automatic directory change.

Directory changes made with CD are recorded in the directory history list and can be displayed in the directory history window, which allows you to return quickly to a recently-used directory.

CD never changes the default drive. If you change directories on one drive, switch to another drive, and then enter **CD -**, the directory will be restored on the first drive but the current drive will not be changed.

Options

/D	(Drive) Changes the current drive as well as directory (like CDD). This option is included for compatibility with the undocumented CD /D command available in Windows NT 4.0's *CMD.EXE*.

CDD

Purpose: Change the current disk drive and directory.

Format: **CDD [/A /S[drive ...]] [*path* | -]**

path: The name of the directory (or drive and directory) to change to.
drive: A drive or list of drives to include in the extended directory search database.

/A(ll drives) **/S**(earch tree)

See also: CD, MD, PUSHD, RD, CDPATH, and Directory Navigation.

Usage

CDD is similar to the CD command, except that it also changes the default disk drive if one is specified. CDD will change to the directory and drive you name. To change from the root directory on drive A to the subdirectory *C:\WP:*

```
[a:\] cdd c:\wp
[c:\wp]
```

You can change to the parent directory with **CDD ..**; you can also go up one additional directory level with each additional [**.**]. For example, **CDD ....** will go up three levels in the directory tree (see Extended Parent Directory Names).

CDD can also change to a network drive and directory specified with a UNC name.

When you use CDD to change to a directory on an HPFS, NTFS, or LFN drive, you must quote the *path* name if it contains whitespace or special characters. See File Names for additional details.

If CDD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database in order to find a matching directory and switch to it. You can also use wildcards in the *path* to force an extended directory search. See Directory Navigation for complete details on these and other directory navigation features.

CDD saves the current drive and directory before changing to a new directory. You can switch back to the previous drive and directory by entering **CDD -** (there must be a space between the CDD command and the hyphen). You can switch back and forth between two drives and directories by repeatedly entering **CDD -**. The saved directory is the same for both the CD and CDD commands. Drive changes and automatic directory changes also modify the saved directory, so you can use **CDD -** to return to a directory that you exited with a drive change or an automatic directory change.

Directory changes made with CDD are also recorded in the directory history list and can be displayed in the directory history window, which allows you to return quickly to a recently-used directory.

Options

/A (All drives) When CDD is used with this option, it displays the current directory on all drives from C: to the last drive in the system. You cannot move to a new drive and directory and use **/A** in the same command.

/S (Search tree) Builds or rebuilds the Extended Directory Search database, *JPSTREE.IDX*. You cannot move to a new drive and directory and use **/S** in the same command.

To include all local hard drives in the database use the command:

```
cdd /s
```

To limit or add to the list of drives included in the database, list the drives and network volume names after the **/S** switch. For example, to include drives C, D, E, and the network volume *\\server\dir1* in the database, use this command:

```
cdd /s cde \\server\dir1
```

All non-hidden directories on the listed drives will be indexed; you cannot restrict the database to certain directories within a drive. Each time you use **/S**, everything in the previous directory database is replaced by the new database that is created.

CLS

Purpose: Clear the video display and move the cursor to the upper left corner; optionally change the default display colors.

Format: **CLS [/C /S] [[BRIght] *fg* ON [BRIght] *bg***

fg: The new foreground color.
bg: The new background color.

/C(lear buffer) **/S**(croll buffer)

Usage

CLS can be used to clear the screen without changing colors, or to clear the screen and change the screen colors simultaneously. These two examples show how to clear the screen to the default colors, and to bright white letters on a blue background:

```
[c:\] cls
[c:\] cls bright white on blue
```

CLS is often used in batch files to clear the screen before displaying text.

See Colors and Color Names for details about colors.

Options

/C (Clear buffer) Clear the entire scrollback buffer. If **/C** is not used, only the visible portion of the 4NT screen is cleared.

/S (Scroll buffer) Clear the screen by scrolling the buffer, rather than filling the screen with blanks (the default method). This preserves the text on the screen in the scrollback buffer if it is larger than the visible window. This switch may not give correct results when the buffer size is less than twice the window size.

COLOR

Purpose: Change the default display colors.

Format: **COLOR [BRIght] *fg* ON [BRIght] bg**

fg: The new foreground color.
bg: The new background color.

See also: CLS, and Colors and Color Names for details about using colors.

Usage

COLOR is normally used in batch files before displaying text. For example, to set screen colors to bright white on blue, you can use this command:

```
[c:\] color bright white on blue
```

4NT also supports the same syntax as the version of *CMD.EXE* that is included with Windows NT 4.0 and later:

COLOR bf

In this syntax, **b** is a hexadecimal digit that specifies the background color and **f** is a hexadecimal digit that specifies the foreground color. See your Windows NT documentation for more information.

COPY

Purpose: Copy data between disks, directories, files, or physical hardware devices (such as your printer or serial port).

Format: **COPY [/A:[[-]rhsda] /C /E /H /K /M /N /P /Q /R /S /T /U /V /X /Z]** ***source*** **[+] ... [/A /B]** ***destination*** **[/A/B]**

source: A file or list of files or a device to copy ***from***.
destination: A file, directory, or device to copy ***to***.

/A(SCII)	**/P**(rompt)
/A: (Attribute select)	**/Q**(uiet)
/B(inary)	**/R**(eplace)
/C(hanged)	**/S**(ubdirectories)
/E (no error messages)	**/T**(otals)
/H(idden)	**/U**(pdate)
/K(eep attributes)	**/V**(erify)
/M(odified)	**/X** (clear archive)
/N(othing)	**/Z** (overwrite)

See also: ATTRIB, MOVE, and REN.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, size, or exclude ranges anywhere on the line apply to all *source* files.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

The COPY command accepts all traditional syntax and options and adds many new features.

The simplest use of COPY is to make a copy of a file, like this example which makes a copy of a file called *FILE1.ABC*:

```
[c:\] copy file1.abc file2.def
```

You can also copy a file to another drive and/or directory. The following command copies *FILE1* to the \ *MYDIR* directory on drive E:

```
[c:\] copy file1 e:\mydir
```

When you COPY files to or from an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names for additional details.

Copying Files

You can copy several files at once by using wildcards:

```
[c:\] copy *.txt e:\mydir
```

You can also list several *source* files in one command. The following command copies 3 files from the current directory to the *\MYDIR* directory on drive E:

```
        [c:\] copy file1 file2 file3 e:\mydir
```

COPY also understands include lists, so you can specify several different kinds of files in the same command. This command copies the *.TXT*, *.DOC*, and *.BAT* files from the *E:\MYDIR* directory to the root directory of drive A:

```
        [c:\] copy e:\mydir\*.txt;*.doc;*.bat a:\
```

If there is only one argument on the line, COPY assumes it is the *source*, and uses the current drive and directory as the *destination*. For example, the following command copies all the *.DAT* files on drive A to the current directory on drive C:

```
        [c:\data] copy a:*.dat
```

If there are two or more arguments on the line, separated by spaces, then COPY assumes that the last argument is the *destination* and copies all *source* files to this new location. If the *destination* is a drive, directory, or device name then the *source* files are copied individually to the new location. If the *destination* is a file name, the first *source* file is copied to the *destination*, and any additional *source* files are then appended to the new *destination* file.

For example, the first of these commands copies the *.DAT* files from the current directory on drive A individually to *C:\MYDIR* (which must already exist as a directory); the second appends all the *.DAT* files together into one large file called *C:\DATA* (assuming *C:\DATA* is not a directory):

```
        [c:\] copy a:*.dat c:\mydir\
        [c:\] copy a:*.dat c:\data
```

When you copy to a directory, if you add a backslash [****] to the end of the name as shown in the first example above, COPY will display an error message if the name does not refer to an existing directory. You can use this feature to keep COPY from treating a mistyped *destination* directory name as a file name and attempting to append all your *source* files to a *destination* **file**, when you really meant to copy them individually to a *destination* **directory**.

To copy a file to a device such as the printer, use the device name as the ***destination***, for example:

```
        [c:\] copy schedule.txt prn
```

To copy text to or from the clipboard use **CLIP:** as the device name. Using CLIP: with non-text data will produce unpredictable results. See Redirection and Piping for additional details.

Appending Files

A plus [**+**] tells COPY to append two or more files to a single *destination* file. If you list several *source* files separated with [**+**] and don't specify a *destination*, COPY will use the name of the first *source* file as the destination, and append each subsequent file to the first file.

For example, the following command will append the contents of *C:\MEMO2* and *C:\MEMO3* to *C:\MEMO1* and leave the combined contents in the file named *C:\MEMO1*:

```
        [c:\] copy memo1+memo2+memo3
```

To append the same three files but store the result in *BIGMEMO:*

```
        [c:\] copy memo1+memo2+memo3 bigmemo
```

If no *destination* is specified, the destination file will always be created in the current directory even if the first *source* file is in another directory or on another drive.   For example, this command willappend *C:\MEM\MEMO2* and *C:\MEM\MEMO3* to *D:\DATA\MEMO1*, and leave the result in *C:\MEM\MEMO1*:

```
        [c:\mem] copy d:\data\memo1+memo2+memo3
```

You cannot append files to a device (such as a printer); if you try to do so, COPY will ignore the [**+**] signs and copy the files individually.   If you attempt to append several *source* files to a *destination* directory or disk, COPY will append the files and place the copy in the new location with the same name as the first *source* file.

Advanced Features

If your *destination* has wildcards in it, COPY will attempt to match them with the *source* names.   For example, this command copies the *.DAT* files from drive A to *C:\MYDIR* and gives the new copies the extension *.DX*:

```
        [c:\] copy a:*.dat c:\mydir\*.dx
```

This feature can give you unexpected results if you use it with multiple *source* file names.   For example, suppose that drive A contains *XYZ.DAT* and *XYZ.TXT*.   The command

```
        [c:\] copy a:\*.dat a:\*.txt c:\mydir\*.dx
```

will copy *A:XYZ.DAT* to *C:\MYDIR\XYZ.DX*.   Then it will copy *A:XYZ.TXT* to *C:\MYDIR\XYZ.DX*, overwriting the first file it copied.

COPY also understands include lists, so you can specify several different kinds of files in the same command.   This command copies the *.TXT*, *.DOC*, and *.BAT* files from the *E:\MYDIR* directory to the root directory of drive A:

```
        [c:\] copy e:\mydir\*.txt;*.doc;*.bat a:\
```

You can use date, time, and size ranges to further define the files that you want to copy.   This example copies every file in the *E:\MYDIR* directory, which was created or modified yesterday, and which is also 10,000 bytes or smaller in size, to the root directory of drive A:

```
        [c:\] copy /[d-1] /[s0,10000] e:\mydir\*.* a:\
```

You can also use file exclusion ranges to restrict the list of files that would normally be selected with wildcards.   This example copies every file in the *E:\MYDIR* directory except backup (*.BAK* or *.BK!*) files:

```
        [c:\] copy /[!*.bak;*.bk!] e:\mydir\*.* a:\
```

COPY will normally process *source* files which do not have the hidden or system attribute, and will ignore the read-only and archive attributes.   It will always set the archive attribute and clear the read-only attribute of *destination* files.   In addition, if the *destination* is an existing file with the read-only attribute, COPY will generate an "Access Denied" error and refuse to overwrite the file.   You can alter some of these behaviors with switches:

/A: Forces COPY to process *source* files with the attributes you specify after the "**:**", or to process all *source* files regardless of attributes (if **/A:** is used by itself).

/H   Forces COPY to process hidden and system *source* files, as well as normal files.   The hidden and system attributes from each source file will be preserved when creating the *destination* files.

/K Retains the read-only attribute from each *source* file when creating the *destination* file. See **/K** below for a special note if you are running under Novell Netware.

/Z Forces COPY to overwrite an existing read-only *destination* file.

Use caution with **/A:**, **/H**, or **/K** when both the *source* and *destination* directories contain file descriptions. If the *source* file specification matches the description file name (normally *DESCRIPT.ION*), and you use a switch which tells COPY to process hidden files, the *DESCRIPT.ION* file itself will be copied, overwriting any existing file descriptions in the *destination* directory. For example, if the *\DATA* directory contains file descriptions this command would overwrite any existing descriptions in the *\SAVE* directory:

```
[c:\data] copy /h d*.* \save\
```

(If you remove the hidden attribute from the *DESCRIPT.ION* file the same caution applies even if you do not use **/A:**, **/H**, or **/K**, as *DESCRIPT.ION* is then treated like any other file.)

Options

The **/A**(SCII) and **/B**(inary) options apply to the preceding filename and to all subsequent filenames on the command line until the file name preceding the next **/A** or **/B**, if any. The other options (/**A:**, **/C**, **/E**, **/H**, **/K**, **/M**, **/N**, **/P**, **/Q**, **/R**, **/S**, **/T**, **/U**, **/V**, **/X**, **/Z**) apply to all filenames on the command line, no matter where you put them. For example, either of the following commands could be used to copy a font file to the printer in binary mode:

```
[c:\] copy /b myfont.dat prn
[c:\] copy myfont.dat /b prn
```

Some options do not make sense in certain contexts, in which case COPY will ignore them. For example, you cannot prompt before replacing an existing file when the *destination* is a device such as the printer -- there's no such thing as an "existing file" on the printer. If you use conflicting output options, like **/Q** and **/P**, COPY will generally take a "conservative" approach and give priority to the option which generates more prompts or more information.

/A (ASCII) If you use **/A** with a *source* filename, the file will be copied up to, but not including, the first Ctrl-Z (Control-Z or ASCII 26) character in the file (some application programs use the Ctrl-Z to mark the end of a file). If you use **/A** with a *destination* filename, a Ctrl-Z will be added to the end of the file. **/A** is the default when appending files, or when the *destination* is a device like NUL or PRN, rather than a disk file. Also see **/B**.

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. You **must** include the colon [**:**] with this option to distinguish it from the **/A**(SCII) switch, above. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **COPY /A: ...**), COPY will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

See the cautionary note under **Advanced Features** above before using **/A:** when both *source* and *destination* directories contain file descriptions.

/B (Binary) If you use **/B** with a *source* filename, the entire file is copied; Ctrl-Z characters in the file do not affect the copy operation. Using **/B** with a *destination* filename prevents addition of a Ctrl-Z to the end of the *destination* file. **/B** is the default for normal file copies. Also see **/A**.

/C (Changed files) Copy files only if the *destination* file exists and is older than the *source* (see also **/U**). This option is useful for updating the files in one directory from those in another without copying any newly created files.

/E (no Error messages) Suppress all non-fatal error messages, such as "File not found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/H (Hidden) Copy all matching files including those with the hidden and/or system attribute set. See the cautionary note under **Advanced Features** above before using **/H** when both *source* and *destination* directories contain file descriptions.

/K (Keep attributes) To maintain compatibility with *CMD.EXE*, and Netware, COPY normally maintains the hidden and system attributes, sets the archive attribute, and removes the read-only attribute on the target file. **/K** tells COPY to also maintain the read-only attribute on the *destination* file. However, if the *destination* is on a Novell Netware volume, this option will fail to maintain the read-only attribute. This is due to the way Netware handles file attributes, and is not a problem in COPY.

/M (Modified) Copy only those files with the archive attribute set, *i.e.*, those which have been modified since the last backup. The archive attribute of the *source* file will **not** be cleared after copying; to clear it, use the **/X** switch, or use the ATTRIB command.

/N (Nothing) Do everything except actually perform the copy. This option is useful for testing what the result of a complex COPY command will be. **/N** does **not** prevent creation of *destination* subdirectories when it is used with **/S**.

/P (Prompt) Ask the user to confirm each *source* file. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames or the total number of files copied. This option is most often used in batch files. See also **/T**.

/R (Replace) Prompt the user before overwriting an existing file. Your options at the prompt are explained in detail under Page and File Prompts.

/S (Subdirectories) Copy the subdirectory tree starting with the files in the *source* directory plus each subdirectory below that. The *destination* must be a directory; if it doesn't exist, COPY will attempt to create it. COPY will also attempt to create needed subdirectories on the tree below the *destination*, including empty *source* directories. COPY /S creates one or more destination directories, they will be added automatically to the extended directory search database.

If you attempt to use COPY /S to copy a subdirectory tree into part of itself, COPY will detect the resulting infinite loop, display an error message, and exit.

/T (Totals) Turns off the display of filenames, like **/Q**, but does display the total number of files copied.

/U (Update) Copy each *source* file only if it is newer than a matching *destination* file or if a matching *destination* file does not exist (see also **/C**). This option is useful for keeping one directory matched with another with a minimum of copying.

/V (Verify) Verify each disk write. This is the same as executing the VERIFY ON command, but is only active during the COPY. **/V** does **not** read back the file and compare its contents with what was written; it only verifies that the data written to disk is physically readable.

/X Clears the archive attribute from the source file after a successful copy. This option is most useful if you are using COPY to maintain a set of backup files.

/Z Overwrites read-only *destination* files. Without this option, COPY will fail with an "Access denied" error if the *destination* file has its read-only attribute set. This option allows COPY to overwrite read-only files without generating any errors.

DATE

Purpose: Display and optionally change the system date.

Format: **DATE [/T] [*mm* -*dd* -*yy*]**

mm: The month (1 - 12).
dd: The day (1 - 31).
yy: The year (00 - 99, or a 4- digit year).

/T (Display only)

See also: TIME.

Usage

If you simply type DATE without any parameters, you will see the current system date and time, and be prompted for a new date. Press ENTER if you don't wish to change the date. If you type a new date, it will become the current system date, which is included in the directory entry for each file as it is created or altered:

```
[c:\] date
Mon  Dec 22, 1997  9:30:06
Enter new date (mm-dd-yy):
```

You can also enter a new system date by typing the DATE command plus the new date on the command line:

```
[c:\] date 10-16-97
```

You can use hyphens, slashes, or periods to separate the month, day, and year entries. The year can be entered as a 2-digit or 4-digit value. Two-digit years between 80 and 99 are interpreted as 1980 - 1999; values between 00 and 79 are interpreted as 2000 - 2079.

DATE adjusts the format it expects depending on your country settings. When entering the date, use the correct format for the country setting currently in effect on your system.

Options

/T: (Display only) Displays the current date but does not prompt you for a new date. If a new date is specified in the same command as **/T**, the new date will be ignored.

DEL / ERASE

Purpose: Erase one file, a group of files, or entire subdirectories.

Format: **DEL [/A:[[-]rhsda] /E /N /P /Q /S /T /W /X /Y /Z] *file*...**

or

ERASE [/A:[[-]rhsda] /E /N /P /Q /S /T /W /X /Y /Z] *file*...

file: The file, subdirectory, or list of files or subdirectories to erase.

/A: (Attribute select)	**/T**(otal)
/E (No error messages)	**/W**(ipe)
/N(othing)	**/X** (remove empty subdirectories)
/P(rompt)	**/Y**(es to all prompts)
/Q(uiet)	**/Z**(ap hidden and read-only files)
/S(ubdirectories)	

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

DEL and ERASE are synonyms, you can use either one.

Use the DEL and ERASE commands with caution; the files and subdirectories that you erase may be impossible to recover without specialized utilities and a lot of work.

To erase a single file, simply enter the file name:

```
[c:\] del letters.txt
```

You can also erase multiple files in a single command. For example, to erase all the files in the current directory with a *.BAK* or *.PRN* extension:

```
[c:\] del *.bak *.prn
```

When you use DEL on an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names for additional details.

To exclude files from a DEL command, use a file exclusion range. For example, to delete all files in the current directory except those whose extension is *.TXT*, use a command like this:

```
[c:\] del /[!*.TXT] *.*
```

When using exclusion ranges or other more complex options you may want to use the **/N** switch first, to preview the effects of the DEL without actually deleting any files.

If you enter a subdirectory name, or a filename composed only of wildcards (***** and / or **?**), DEL asks for confirmation (**Y** or **N**) unless you specified the **/Y** option. If you respond with a **Y**, DEL will delete all the files in that subdirectory (hidden, system, and read-only files are only deleted if you use the **/Z** option).

Use caution when using wildcards with DEL. For compatibility with *CMD.EXE*, 4NT's wildcard matching will match both short and long filenames. This can delete files you did not expect; see LFN File

Searches for additional details.

DEL displays the amount of disk space recovered, unless the **/Q** option is used (see below). It does so by comparing the amount of free disk space before and after the DEL command is executed. This amount may be incorrect if you are using a deletion tracking system which stores deleted files in a hidden directory, or if, under a multitasking system, another program performs a file operation while the DEL command is executing.

Remember that DEL removes file descriptions along with files. Most deletion tracking systems will not be able to save or recover a file's description, even if they can save or recover the data in a file.

When a file is deleted, its disk space is returned to the operating system for use by other files. However, the contents of the file remain on the disk until they are overwritten by another file. If you wish to obliterate a file or wipe its contents clean, use DEL /W, which overwrites the file with zeros before deleting it. Use this option with caution — once a file is obliterated, it is impossible to recover.

DEL returns a non-zero exit code if no files are deleted, or if another error occurs. You can test this exit code with the %_? environment variable, and use it with conditional commands (**&&** and **||**).

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **DEL /A: ...**), DEL will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/N (Nothing) Do everything except actually delete the file(s). This is useful for testing what the result of a DEL would be.

/P (Prompt) Prompt the user to confirm each erasure. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames as they are deleted, or the number of files deleted or bytes freed. See also **/T**.

/S (Subdirectories) Delete the specified files in this directory and all of its subdirectories. This can be used to delete all the files in a subdirectory tree or even a whole disk. It should be used with caution!

/T (Total) Don't display filenames as they are deleted, but display the total number of files deleted plus the amount of free disk space recovered.

/W (Wipe) Clear the file to zeros before deleting it. Use this option to completely obliterate a file's contents from your disk. Once you have used this option it is impossible to recover the file even if you are using an undelete utility, because the contents of the file are destroyed before it is deleted. **/W** overwrites the file only once; it does **not** adhere to security standards which require multiple overwrites with varying data when destroying sensitive information.

/X (Remove empty subdirectories) Remove empty subdirectories after deleting (only useful when used with **/S**). If DEL deletes one or more directories, they will be removed automatically from the extended directory search database.

/Y (Yes) The reverse of **/P** -- it assumes a **Y** response to everything, including deleting an entire subdirectory tree. 4NT normally prompts before deleting files when the name consists only of wildcards or a subdirectory name (see above); **/Y** overrides this protection, and should be used with extreme caution!

/Z (Zap) Delete read-only, hidden, and system files as well as normal files. Files with the read-only, hidden, or system attribute set are normally protected from deletion; **/Z** overrides this protection, and should be used with caution. Because EXCEPT works by hiding files, **/Z** will override an EXCEPT command. However, files specified in a file exclusion range will not be deleted by DEL /Z.

For example, to delete the entire subdirectory tree starting with *C:\UTIL*, including hidden and read-only files, without prompting (use this command with CAUTION!):

```
[c:\] del /sxyz c:\util\
```

DELAY

Purpose: Pause for a specified length of time.

Format: **DELAY [*seconds*]**

seconds: The number of seconds to delay.

Usage

DELAY is useful in batch file loops while waiting for something to occur. To wait for 10 seconds:

```
delay 10
```

DELAY is most useful when you need to wait a specific amount of time for an external event, or check a system condition periodically. For example, this batch file checks the battery status (as reported by your Advanced Power Management drivers) every 15 seconds, and gives a warning when battery life falls below 30%:

```
do forever
   iff %_apmlife lt 30 then
      beep 440 4 880 4 440 4 880 4
      echo Low Battery!!
   endiff
   delay 15
enddo
```

The ***seconds*** value can be as large as 1 billion seconds (34 years!).

For delays shorter than one second, use the BEEP command with an inaudible frequency (below 20 Hz).

4NT uses the minimum possible processor time during a DELAY, in order to allow other applications full use of system resources.

You can cancel a delay by pressing **Ctrl-C** or **Ctrl-Break**.

DESCRIBE

Purpose: Create, modify, or delete file and subdirectory descriptions.

Format: **DESCRIBE [/A:[[-]rhsda]] *file* [[/D]*"description"*] ...**

file: The file, directory, or list of files and directories to operate on.
"description": The description to attach to the file.

/A: (Attribute select) **/D**(escription follows)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

DESCRIBE adds descriptions to files and subdirectories. The descriptions can be displayed by DIR in single-column mode, and by SELECT. Descriptions let you identify your files in much more meaningful ways than you can in an eight-character filename.

You enter a description on the command line by typing the DESCRIBE command, the filename, and the description in quotation marks, like this:

```
[c:\] describe memo.txt "Memo to Bob about party"
```

If you don't put a description on the command line, DESCRIBE will prompt you for it:

```
[c:\] describe memo.txt
Describe "memo.txt" : Memo to Bob about party
```

If you use wildcards or multiple filenames with the DESCRIBE command and don't include the description text, you will be prompted to enter a description for each file. If you do include the description on the command line, all matching files will be given the same description.

If you use DESCRIBE on an HPFS, NTFS, or LFN drive, you must quote the ***file*** name if it contains whitespace or special characters. See File Names for additional details.

If you enter a quoted description on the command line, and the text matches the name of a file in the current directory, the command processor will treat the string as a quoted file name, not as description text as you intended. To resolve this problem use the **/D** switch immediately prior to the quoted description (with no intervening spaces). For example, if the current directory contains the files *DATA.TST* and *"Test File"*, the first of these commands will work as intended, but the second will not (in the second example the string "test file" will be treated as a second file name, when it is intended to be description text):

```
[c:\] describe data.tst /D"test file"
[c:\] describe data.tst "test file"
```

On drives which support long file names you will not see file descriptions in a normal DIR display, because DIR must leave space for the long filenames. To view the descriptions, use **DIR /Z** to display the directory in FAT format.

Each description can be up to 511 characters long. You can change this limit with the DescriptionMax directive in *4NT.INI*. In order to fit your descriptions on a single line in a standard DIR display, keep them to 40 characters or less (longer descriptions are wrapped in the DIR output). DESCRIBE can edit descriptions longer than DescriptionMax (up to a limit of 511 characters), but will not allow you to lengthen the existing text.

The descriptions are stored in each directory in a hidden file called *DESCRIPT.ION*. Use the ATTRIB command to remove the hidden attribute from this file if you need to copy or delete it. (*DESCRIPT.ION* is always created as a hidden file, but will not be re-hidden by 4NT if you remove the hidden attribute.)

You can change the description file name with the SETDOS /D command, or the DescriptionName directive in the *4NT.INI* file, and retrieve it with the %_DNAME internal variable. Use caution when changing the description file name, as changing the name from the default will make it difficult to transfer file descriptions to another system.

The description file is modified appropriately whenever you perform an internal command which affects it (such as COPY, MOVE, DEL, or RENAME), but not if you use an external program (such as XCOPY or a visual shell). You can disable description processing on the Options 1 page of the OPTION command, with the Descriptions directive in the *.INI* file, or with SETDOS /D.

When you COPY or MOVE files between two directories, both of which have descriptions, and you use switches which enable processing of hidden files (or you have removed the hidden attribute from *DESCRIPT.ION*), you must use caution to avoid overwriting existing file descriptions in the *destination* directory with the *DESCRIPT.ION* file from the *source* directory. See the notes under the **Advanced Features** sections of COPY and MOVE for additional details.

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **DESCRIBE /A: ...**), DESCRIBE will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/D (Description follows): The quoted string immediately following this switch is a description, not a file name. Use **/D** to avoid any ambiguity in the meaning of quoted strings. See the **Usage** section above for details.

DETACH

Purpose: Start a Windows NT program in detached mode.

Format: **DETACH *command***

command: The name of a command to execute, including an optional drive and path specification.

See also: START.

Usage

When you start a program with DETACH, that program cannot use the keyboard, mouse, or video display. It is "detached" from the normal means of user input and output. However, you can redirect the program's standard I/O to other devices if necessary, using redirection symbols.

The **command** can be an internal command, external command, alias, or batch file. If it is not an external command, 4NT will detach a copy of itself to execute the command.

For example, the following command will detach a copy of 4NT to run the batch file *XYZ.BTM*:

```
[c:\] detach xyz.btm
```

Once the program has started, 4NT returns to the prompt immediately. It does not wait for a detached program to finish.

There is no standard way to stop a detached program. If the program does not terminate on its own you must reboot the system or use an appropriate task manager or external utility to stop it.

DIR

Purpose: Display information about files and subdirectories.

Format: **DIR [/1 /2 /4 /A[[:][-]rhsda] /B /C /D /E /F /G /H /I"text" /J /K /L /M /N /O[[:][-]adeginrsu] /P /R /S /T[:acw]/U /V /W /Z] [*file*...]**

file: The file, directory, or list of files or directories to display.

/1 (one column)	**/L**(ower case)
/2 (two columns)	**/M** (suppress footer)
/4 (four columns)	**/N**(ew format)
/A(ttribute select)	**/O**(rder)
/B(are)	**/P**(ause)
/C(ompression)	**/R** (disable wRap)
/D(isable color coding)	**/S**(ubdirectories)
/E (upper case)	**/T** (aTtribute) or (Time)
/F(ull path)	**/U** (sUmmary information)
/G (allocated size)	**/V**(ertical sort)
/H(ide dots)	**/W**(ide)
/I (match descriptions)	**/X** (display short names)
/J(ustify names)	**/Z** (use FAT format)
/K (suppress header)	

See also: ATTRIB, DESCRIBE, SELECT, and SETDOS.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

DIR can be used to display information about files from one or more of your disk directories, in a wide range of formats. Depending on the options chosen, you can display the file name, attributes, and size; the time and date of the last change to the file; the file description; and the file's compression ratio. You can also display information in 1, 2, 4, 5, or more, sort the files several different ways, use color to distinguish file types, and pause after each full screen.

The various DIR displays are controlled through options or switches. The best way to learn how to use the many options available with the DIR command is to experiment. You will soon know which options you want to use regularly. You can select those options permanently by using the ALIAS command.

For example, to display all the files in the current directory, in 2 columns, sorted vertically (down one column then down the next), and with a pause at the end of each page:

```
[c:\] dir /2/p/v
```

To set up this format as the default, using an alias:

```
[c:\] alias dir=*dir /2/p/v
```

When you use DIR on an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names for additional details.

The following sections group DIR's features together in several categories. Many of the sections move from a general discussion to more technical material.   If you find some of the information in a category too detailed for your needs, feel free to skip to the next section.   The sections are:

Selecting Files

Default DIR Output Format

Switching Formats

Multiple Column Displays

Color-Coded Directories

Redirected Output

Other Notes

Options

Selecting Files

DIR can display information about a single file or about several, dozens, hundreds, or thousands of files at once.   To display information about a single file, just add the name of the file to the DIR command line:

```
[c:\] dir january.wks
```

The simplest way to view information about several files at once is to use wildcards.   DIR can work with traditional wildcard characters (* and ?) and extended wildcards.   For example to display all of the *.WKS* files in the current directory:

```
[c:\] dir *.wks
```

To display all *.TXT* files whose names begin with A, B, or C:

```
[c:\] dir [abc]*.txt
```

If you don't specify a filename, DIR defaults to ***.*** on traditional FAT drives, and ***** on HPFS, NTFS, and LFN drives.   This default displays all non-hidden files and subdirectories in the current directory.

If you link two or more filenames together with spaces, DIR will display all of the files that match the first name and then all of the files that match the second name.   You may use a different drive and path for each filename.   This example lists all of the *.WKS* and then all of the *.WK1* files in the current directory:

```
[c:\] dir *.wks *.wk1
```

If you use an include list to link multiple filenames, DIR will display the matching filenames in a single listing.   Only the first filename in an include list can have a path; the other files must be in the same path.   This example displays the same files as the previous example, but the *.WKS* and *.WK1* files are intermixed:

```
[c:\] dir *.wks;*.wk1
```

You can include files in the current or named directory plus all of its subdirectories by using the **/S** option.   This example displays all of the *.WKS* and *.WK1* files in the D:\DATA directory and each of its subdirectories:

```
[c:\] dir /s d:\data\*.wks;*.wk1
```

You can also select files by their attributes by using the **/A** option.   For example, this command displays the names of all of the subdirectories of the current directory:

```
[c:\] dir /a:d
```

Finally, with the **/I** option, DIR can select files to display based on their descriptions (see the DESCRIBE command for more information on file descriptions). DIR will display a file if its description matches the text after the **/I** switch. The search is not case sensitive. You can use wildcards and extended wildcards as part of the text. For example, to display any file described as a "Test File" you can use this command:

```
[c:\] dir /i"test file"
```

If you want to display files that include the words "test file" anywhere in their descriptions, use extended wildcards like this:

```
[c:\] dir /i"*test file*"
```

To display only those files which do **not** have descriptions, use:

```
[c:\] dir /I"[]"
```

In addition, you can use ranges to select or exclude specific sets of files. For example, to display all files modified in the last week, all files except those with a *.BAK* extension, and all files over 500 KB in size:

```
[c:\] dir /[d-7]
[c:\] dir /[!*.bak]
[c:\] dir /[s500K]
```

You can, of course, mix any of these file selection techniques in whatever ways suit your needs.

Default DIR Output Format

DIR's output varies based on the type of volume or drive on which the files are stored. If the volume supports long file names (HPFS volumes, NTFS volumes, and VFAT volumes under Windows NT), the default DIR format contains 4 columns: the date of the last file modification or write, the time of last write, the file size in bytes, and the file name. The name is displayed as it is stored on the disk, in upper, lower, or mixed case. DIR will wrap filenames from one line to the next if they are too long to fit the width of the display. The standard output format is:

```
 Volume in drive C is C - BOOTUP    Serial ...
 Directory of  C:\4DOS60\*.*

10-24-96  12:17         <DIR>    .
10-24-96  12:17         <DIR>    ..
10-28-96   7:57             967  4dos 6.pif
10-21-96  18:08         212,854  4DOS.COM
11-02-96  10:08              45  4DOS.INI
```

(See **Switching Formats** below for information on changing the standard long filename format to allow room for file descriptions.)

On FAT volumes which do not support long file names, the default DIR format contains 5 columns: the file name, the file size in bytes, the date of the last write, the time of the last write, and the file's description. File names are listed in lower-case; directory names in upper case:

```
 Volume in drive C is C - BOOTUP    Serial ...
 Directory of  C:\4DOS60\*.*
```

```
.             <DIR>     10-24-96  12:17
..            <DIR>     10-24-96  12:17
TEST          <DIR>     11-01-96  16:21
4dos6.pif          967  10-28-96   7:57 4DOS PIF file
4dos.com        212854  10-21-96  18:08 4DOS exe ...
4dos.ini            45  11-02-96  10:08 4DOS conf ...
```

DIR's output is normally sorted by name, with directories listed first. You can change the sort order with the **/O** option. For example, these two commands sort the output by date — the first command lists the oldest file first; the second command lists the oldest file last:

```
[c:\] dir /o:d
[c:\] dir /o:-d
```

When displaying file descriptions, DIR wraps long lines to fit on the screen. DIR displays a maximum of 40 characters of text in each line of a description, unless your screen width allows a wider display. If you disable description wrapping with the **/R** option, the description is truncated at the right edge of the screen, and a right arrow is added at the end of the line to alert you to the existence of additional description text.

Regardless of the volume type, DIR's default output is sorted. It displays directory names first, with "<DIR>" inserted instead of a file size, and then filenames. DIR assumes that sequences of digits should be sorted numerically (for example, the file *DRAW2* is listed before *DRAW03* because 2 is numerically smaller than 03), rather than strictly alphabetically (where *DRAW2* would come second because "2" is after "0" in alphanumeric order). You can change the sort order with the **/O** option. When DIR displays file names in a multi-column format, it sorts file names horizontally unless you use the **/V** option to display vertically sorted output.

DIR's display can be modified in many ways to meet different needs. Most of the following sections describes the various ways you can change DIR's output format.

Switching Formats

On volumes which support long file names, you can force DIR to use a FAT-like format (file name first, followed by file information) with the **/Z** option. If necessary, DIR /Z truncates long file names on LFN, HPFS, and NTFS drives, and adds a right arrow to show that the name contains additional characters.

The standard LFN output format does not provide enough space to show descriptions along with file names. Therefore, if you wish to view file descriptions as part of the DIR listing on a volume which supports long file names, you **must** use the **/Z** option.

4NT will display the alternate, short file names for files with long file names if you use the **/X** option. Used alone, **/X** causes DIR to display names in 2 columns after the size, time, and date: one column for alternate or short file names and the other for long file names. If a file does not have a short or alternate name which is different from the long filename, the first filename column is empty.

If you use **/X** and **/Z** together, DIR will display the short or alternate file names in the FAT-style display format.

If you use the **/B** option, DIR displays just file names and omits the file size, time stamp, and description for each file, for example:

```
[c:\] dir w* /b

WINDOWS
```

```
WINNT
win311
WINALIAS
WINENV.BTM
.....
```

There are several ways to modify the display produced by **/B**. The **/F** option is similar to **/B**, but displays the full path and name of each file, instead of just its name. To view the same information for a directory and its subdirectories use **/B /S** or **/F /S**. You can use **/B /X** to display the short name of each file, with no additional information. To display the short version of both the path and file name for each file, use **/F /X**. For example:

```
[c:\] dir /x/f/s *.pif

C:\MACH64\INSTALL.PIF
C:\PROGRA~1\WINZIP\WZ.PIF
C:\WINDOWS\DOSPRMPT.PIF
C:\WINDOWS\STARTM~1\APPS&U~1\INFOSE~1.PIF
C:\WINDOWS\STARTM~1\PROMPTS\4DOS.PIF
C:\WINDOWS\STARTM~1\PROMPTS\TOCP.PIF
C:\WINDOWS\STARTM~1\PROMPTS\SPECIAL\4DOS(R~1.PIF
.....
```

Multiple Column Displays

DIR has three options, **/2**, **/4**, and **/W**, that create multi-column displays.

The **/2** option creates a 2-column display. On drives which support long filenames, only the name of each file is displayed, with directory names placed in square brackets to distinguish them from file names. On drives which do not support long filenames, or when **/Z** or **/X** is used (see below), the display includes the name, file size, and time stamp for each file.

The **/4** option is similar to **/2**, but displays directory information in 4 columns. On drives which do not support long filenames, or when **/Z** or **/X** is used (see below), the display shows the file name and the file size in kilobytes (KB) or megabytes (MB), with "<D>" in the size column for directories.

The **/W** option displays directory information in 5 or more columns, depending on your screen width. Each entry in a **DIR /W** display contains either the name of a file or the name of a directory. Directory names are placed in square brackets to distinguish them from file names.

If you use one of these options on a drive that supports long file names, and do not select an alternate display format with **/Z** or **/X**, the actual number of columns will be based on the longest name to be displayed and your screen width, and may be less than the number you requested (for example, you might see only three columns even though you used **/4**). If the longest name is too long to fit in on a single line the display will be reduced to one column, and each name will be wrapped, with "extra" blank lines added so that each name takes the same number of lines.

On LFN drives you can use **/Z** with any of the multi-column options to create a traditional FAT-format display, with long names truncated to fit in the available space. If you use **/X**, the traditional FAT-format display is also used, but short names are displayed (rather than truncated long names). The following table summarizes the effects of different options on LFN drives:

		Display Columns	
Format	Normal	/2 or /4	/W

Normal	1 column, long names plus size, date, time	2 - 4 columns, long names only	No. of columns based on longest name, long names only
/Z	1 column, truncated long names plus size, date, time	2 - 4 columns, truncated long names plus other info	5+ columns, truncated long names only
/X	1 column, both names plus size, date, time	2 - 4 columns, short names plus other info	5+ columns, short names only
/Z /X	1 column, short names plus size, date, time	(Same as /X alone)	(Same as /X alone)

Color-Coded Directories

The DIR command can display each file name and the associated file information in a different color, depending on the file's extension.

To choose the display colors, you must either use the SET command to create an environment variable called COLORDIR, or use the Commands page of the OPTION dialogs or a text editor to set the ColorDir directive in your *.INI* file. If you do not use the COLORDIR variable or the ColorDir directive, DIR will use the default screen colors for all files.

If you use both the COLORDIR variable and the ColorDir directive, the environment variable will override the settings in your *.INI* file. You may find it useful to use the COLORDIR variable for experimenting, then to set permanent directory colors with the ColorDir directive.

The format for both the COLORDIR environment variable and the ColorDir directive in the *.INI* file is:

```
ext ... :ColorName; ...
```

where "ext" is a file extension (which may include wildcards) or one of the following file types:

```
DIRS        Directories
RDONLYRead-only files
HIDDENHidden files
SYSTEMSystem files
ARCHIVE     Files modified since the last backup
```

and "ColorName" is any valid color name (see Colors and Color Names).

Unlike most color specifications, the background portion of the color name may be omitted for directory colors. If you don't specify a background color, DIR will use the current screen background color.

For example, to display the *.COM* and *.EXE* files in red on the current background, the *.C* and *.ASM* files in bright cyan on the current background, and the read-only files in blinking green on white (this should be entered on one line):

```
[c:\] set colordir=com exe:red; c asm:bright cyan; rdonly:blink green on
white
```

Extended wildcards can be used in directory color specifications. For example, to display *.BAK*, *.BAX*, and *.BAC* files in red:

```
[c:\] set colordir=BA[KXC]:red
```

Redirected Output

The output of the DIR command, like that of most other internal commands, can be redirected to a file, printer, serial port, or other device. However, you may need to take certain DIR command options into account when you redirect DIR's output.

DIR wraps both long file names and file descriptions at the width of your display. Its redirected output will also wrap at the screen width. Use the **/R** option if you wish to disable wrapping of long descriptions.

If you redirect a color-coded directory to a file, DIR will remove the color data as it sends the directory information to a file. It will usually do the same if you redirect output to a character device such as a printer or serial port. However, it is not always possible for DIR to tell whether or not a device is a character device. If you notice that non-colored lines are being sent to the output device and colored lines are appearing on your screen, you can use the **/D** option to temporarily disable color-coding when you redirect DIR's output.

To redirect DIR output to the clipboard, use CLIP: as the output device name, for example:

```
[c:\] dir *.exe > clip:
```

Other Notes

If you have selected a specific country code for your system, DIR will display the date in the format for that country. The default date format is U.S. (mm-dd-yy). The separator character in the file time will also be affected by the country code. Thousands and decimal separators in numeric displays are affected by the country code, and by the ThousandsChar and DecimalChar settings selected on the Options 1 page of the OPTION dialogs, or in the *.INI* file.

If you are using a disk compression program, you can use the **/C** switch to view the amount of compression achieved for each file. When you do, the compression ratio is displayed instead of the file's description. You can also sort the display by compression ratios with the **/O:c** switch. Details for both switches are in the Options section, below.

Options

Options on the command line apply only to the filenames which follow the option, and options at the end of the line apply to the preceding filename only. This allows you to specify different options for different groups of files, yet retains compatibility with the traditional DIR command when a single filename is specified.

/1 Single column display – display the filename, size, date, and time; also displays the description on drives which do not support long filenames. This is the default. If **/T** is used the attributes are displayed instead of the description; if **/C** or **/O:c** is used the compression ratio is displayed instead of the description. This option is most useful if you wish to override a default **/2**, **/4**, or **/W** setting stored in an alias.

/2 Two column display – display just the name (on LFN, HPFS, and NTFS drives), or display the filename, size, date, and time on other drives. See **Multiple Column Displays** for more details.

/4 Four column display – display just the name (on LFN, HPFS, and NTFS drives); or display the filename and size, in K (kilobytes) or M (megabytes) on other drives, with files between 1 and 9.9 megabytes in size displayed in tenths (*i.e.*, "2.4M"). See **Multiple Column Displays** for more details.

/A (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **DIR /A ...**), DIR will display all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included in the listing. For example, **/A:RHS** will display only those files with all three attributes set.

/B (Bare) Suppress the header and summary lines, and display file or subdirectory names only, in a single column. This option is most useful when you want to redirect a list of names to a file or another program. If you use **/B** with **/S**, DIR will show the full path of each file (the same display as **/F**) instead of simply its name and extension. If you use **/B** with **/X** on an LFN or NTFS drive, DIR will display the short name of each file instead of the long name.

/C (Compression) Display per-file and total compression ratio on compressed drives. The compression ratio is displayed instead of the file description or attributes. The ratio is left blank for directories and files with a length of 0 bytes, and for files on non-compressed drives. **/C** only works in single-column mode; it is ignored if **/2**, **/4**, or **/W** is used. Only compressed NTFS drives are supported.

The numerator of the displayed compression ratio is the amount of space which would be allocated to the file if the compression utility were not in use, based on the compressed drive's cluster size (usually 8K bytes). The denominator is the space actually allocated for the compressed file. For example, if a file is allocated 6,144 bytes when compressed, and would require 8,192 bytes if uncompressed, the displayed compression ratio would be 8,192 / 6,144, or 1.3 to 1.

/D (Disable color coding) Temporarily disable directory color coding. May be required when color-coded directories are used and DIR output is redirected to a character device like the printer (*e.g.*, PRN or LPT1) or serial port (*e.g.*, COM1 or COM2). **/D** is not required when DIR output is redirected to a file.

/E Display filenames in the traditional upper case; also see SETDOS /U and the UpperCase directive in *4NT.INI*.

/F (Full path) Display each filename with its drive letter and path in a single column, without other information. If you use **/F** with **/X** on a volume which supports long filenames, the "short" version of the entire path is displayed.

/G Display the allocated disk space instead of the actual size of each file.

/H (Hide dots) Suppress the display of the "." and ".." directories.

/I Display filenames by matching text in their descriptions. The text can include wildcards and extended wildcards. The search text must be enclosed in quotation marks. You can select all filenames that have a description with **/I"[?]*"**, or all filenames that do not have a description with **/I"[]"**.

The **/I** option may be used to select files even if descriptions are not displayed (for example, if

/2 is used). However, **/I** will be ignored if **/C** or **/O:c** is used.

/J (Justify names) Justify (align) filename extensions and display them in the traditional format.

/K Suppress the header (disk and directory name) display.

/L (Lower case) Display file and directory names in lower case; also see SETDOS /U and the UpperCase directive in *4NT.INI*.

/M Suppress the footer (file and byte count totals) display.

/N (New format) Use the long filename display format, even if the files are stored on a volume which does not support long filenames. See also **/Z**.

/O (Order) Set the sorting order. You may use any combination of the following sorting options; if multiple options are used, the listing will be sorted with the first sort option as the primary key, the next as the secondary key, and so on:

-	Reverse the sort order for the next option.
a	Sort in ASCII order, not numerically, when there are digits in the name.
c	Sort by compression ratio (the least compressed file in the list will be displayed first). For single-column directory displays in the traditional short filename format, the compression ratios will be used as the basis of the sort and will also be displayed. For wider displays (**/2**, **/4**, and **/W**) and displays in LFN format, the compression ratios will be used to determine the order but will not be displayed. For information on supported compression systems see **/C** above.
d	Sort by date and time (oldest first); for drives which support long filenames also see **/T:acw**.
e	Sort by extension.
g	Group subdirectories first, then files.
i	Sort by file description.
n	Sort by filename (this is the default).
r	Reverse the sort order for all options.
s	Sort by size.
u	Unsorted.

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/R (disable wRap) Forces long descriptions to be displayed on a single line, rather than wrapped onto two or more lines. Use **/R** when output is redirected to a character device, such as a serial port or the printer; or when you want descriptions truncated, rather than wrapped, in the on-screen display.

/S (Subdirectories) Display file information from the current directory and all of its subdirectories. DIR will only display headers and summaries for those directories which contain files that match the filename(s), ranges, and attributes that you specify.

/T (aTtribute display) Display the filenames, attributes, and descriptions. The descriptions will be wrapped onto the next line, if necessary, unless you also use the **/R** (truncate) option. If you use both **/T** and **/R**, descriptions are truncated after 34 characters on an 80-column display. The attributes are displayed in the format RHDSA, with the following meanings:

R	Read-only
H	Hidden
D	Subdirectory

S	System
A	Archive

On drives which support long file names, if you wish to add another option after **/T**, you must start the next option with a forward slash. If you don't, the command processor will interpret the **/T** as the **/T:acw** time display switch (see below) and the following character as a valid or invalid time selector. For example:

```
[c:\] dir /tz           incorrect, will display error
[c:\] dir /t/z          correct
```

/T:acw (Time display) Specify which of the date and time fields on a drive which supports long filenames should be displayed and used for sorting:

a	Last access date and time (access time is not saved on LFN volumes).
c	Creation date and time.
w	Last write time (default).

/U (sUmmary information) Only display the number of files, the total file size, and the total amount of disk space used. Information on individual files is not displayed.

/V (Vertical sort) Display the filenames sorted vertically rather than horizontally (use with the **/2**, **/4** or **/W** options).

/W (Wide) Display filenames only, horizontally across the screen. On drives which do not support long filenames, or when used with **/Z** or **/X**, **/W** displays as many columns as it can fit into the command processor window, using 16 characters in each column. Otherwise (*i.e.*, when long filenames are displayed) the number of columns depends on the width of the longest name in the listing. See **Multiple Column Displays** for more details.

/X Display both the short name (8-character name plus 3-character extension) and the long name of each file on an LFN or NTFS drive. In normal single-column output the short name is displayed first, followed by the long name. The short name column is left blank if the short name and long name are the same. **/X** also selects short filenames in the **/2**, **/4**, **/B**, **/W**, and **/Z** displays, and short file and path names in the **/F** display.

/Z Display a directory on an HPFS, NTFS, or LFN drive in the traditional FAT format, with the filename at the left and the description at the right. Long names will be truncated to 12 characters unless **/X** is also used; if the name is longer than 12 characters, it will be followed by a right arrow to show that one or more characters have been truncated.

DIRHISTORY

Purpose: Display, add to, clear, or read the directory history list.

Format: **DIRHISTORY [/A *directory* /F /P /R *filename*]**

directory: The name of a directory to be added to the directory history.
filename: The name of a file containing entries to be added to the directory history.

/A(dd)	**/P**(ause)
/F(ree)	**/R**(ead)

See also: HISTORY.

Usage

Every time you change to a new directory or drive, 4NT records the current directory in an internal directory history list. See Directory History for information on the directory history window, which allows you to use the list to return to a previous directory. Also see Directory Navigation.

The DIRHISTORY command lets you view and manipulate the directory history list directly. If no parameters are entered, DIRHISTORY will display the current directory history list:

```
[c:\] dirhistory
```

With the options explained below, you can clear the list, add new directories to the list without changing to them, save the list in a file, or read a new list from a file.

The number of directories saved in the directory history list depends on the length of each directory name. The list size can be specified at startup from 256 to 32767 characters by using the DirHistory directive in *4NT.INI*.

Your directory history list can be stored either locally (a separate history list for each copy of the command processor) or globally (all copies of the command processor share the same list). See Directory History for a discussion of local and global directory history lists.

You can save the directory history list by redirecting the output of DIRHISTORY to a file. This example saves the history to a file called *DIRHIST* and reads it back again:

```
[c:\] dirhistory > dirhist
    .....
[c:\] dirhistory /r dirhist
```

Because the directory history stores each name only once, you don't have to delete its contents before reading back the file unless you want to delete the directories that were visited by the intervening commands.

If you need to save your directory history at the end of each day's work, you might use commands like this in your *4START.BTM* file:

```
if exist c:\dirhist dirhistory /r c:\dirhist
alias shut*down `dirhistory > c:\dirhist`
```

This restores the previous history list if it exists, then defines an alias which will allow you to save the history before shutting off the system.

Options

/A	(Add) Add a directory to the directory history list.
/F	(Free) Erase all entries in the directory history list.
/P	(Prompt) Wait for a key after displaying each page of the list. Your options at the prompt are explained in Page and File Prompts.
/R	(Read) Read the directory history from the specified file and append it to the list currently held in memory.

DIRS

Purpose: Display the current directory stack.

Format: **DIRS**

See also: PUSHD and POPD, and Directory Navigation.

Usage

The PUSHD command adds the current default drive and directory to the directory stack, a list that 4NT maintains in memory. The POPD command removes the top entry of the directory stack and makes that drive and directory the new default. The DIRS command displays the contents of the directory stack, with the most recent entries on top (*i.e.*, the next POPD will retrieve the first entry that DIRS displays).

The directory stack holds 511 characters, enough for 20 to 40 typical drive and directory entries.

DO

Purpose: Create loops in batch files.

Format: **DO [*n* | FOREVER]**

or

DO *varname* = *start* TO *end* [BY *n*]

or

DO [WHILE | UNTIL] *condition*

or

DO *varname* IN [@]set

commands

[ITERATE]

[LEAVE]

commands

ENDDO

varname: The environment variable that will hold the loop counter, filename, or line from a file.
n, start, end: Integers between 0 and 2,147,483,647 inclusive, or internal variables or variable functions that evaluate to such a value.
condition: A test to determine if the loop should be executed.
set: A set of values for the variable.
commands: One or more commands to execute each time through the loop. If you use multiple commands, they must be separated by command separators or be placed on separate lines.

File Selection

Supports extended wildcards, ranges, and include lists for the *set*.

Usage

DO can only be used in batch files; it cannot be used in aliases.

DO can be used to create 4 different kinds of loops. The first, introduced by **DO n**, is a counted loop. The batch file lines between DO and ENDDO are repeated *n* times. For example:

```
do 5
  beep
enddo
```

You can also specify "forever" for ***n*** if you wish to create an endless loop (you can use LEAVE or GOTO to exit such a loop; see below for details).

The second type of loop is similar to a "for loop" in programming languages like BASIC. DO creates an environment variable, *varname*, and sets it equal to the value *start* (if *varname* already exists in the environment, it will be overwritten). DO then begins the loop process by comparing the value of *varname* with the value of *end.* If *varname* is less than or equal to *end*, DO executes the batch file lines up to the

ENDDO.   Next, DO adds 1 to the value of *varname*, or adds the value *n* if BY *n* is specified, and repeats the compare and execute process until *varname* is greater than *end*. This example displays the even numbers from 2 through 20:

```
do i = 2 to 20 by 2
  echo %i
enddo
```

DO can also count down, rather than up.   If *n* is negative, *varname* will decrease by *n* with each loop, and the loop will stop when *varname* is **less** than *end*.   For example, to display the even numbers from 2 through 20 in reverse order, replace the first line of the example above with:

```
do i = 20 to 2 by -2
```

The third type of loop is called a "while loop" or "until loop." DO evaluates the *condition*, which can be any of the tests supported by the IF command, and executes the lines between DO and ENDDO as long as the condition is true.   The loop ends when the condition becomes false.

WHILE tests the condition at the start of the loop.   Therefore, if the condition is false when the loop starts, the statements within the loop will never be executed, and the batch file will continue with the statement after the ENDDO.

UNTIL tests the condition at the end of the loop.   Therefore, if the condition is false when the loop starts, the statements within the loop will still be executed at least once.

The fourth type of loop executes the lines between DO and ENDDO once for every member of a *set* (this is similar to the set used in the FOR command).   Normally, the *set* is a list of files specified with wildcards.   For example:

```
do x in *.txt
```

will execute the loop once for every *.TXT* file in the current directory; each time through the loop the variable **x** will be set to the name of the next file that matches the file specification.

If, between DO and ENDDO, you create a new file that *could* be included in the list of files, it may or may not appear in an iteration of the DO loop.   Whether the new file appears depends on its physical location in the directory structure, a condition over which 4NT has no control.

You can also execute the loop once for each line of text in a file by placing an [**@**] in front of the file name. If you have a file called *DRIVES.TXT* that contains a list of drives on your computer, one drive name per line, you can execute the loop once for each drive this way:

```
do x in @drives.txt
```

To execute the loop once for each line of text in the clipboard, use **CLIP:** as the file name (*e.g.* DO X IN @CLIP:).   CLIP: will not return any data unless the clipboard contains text.   See Redirection for additional information on CLIP:.

Two special commands, ITERATE and LEAVE, can be used inside a DO / ENDDO loop.   ITERATE ignores the remaining lines inside the loop and returns to the beginning of loop for another iteration (unless DO determines that the loop is finished).   LEAVE exits from the current DO loop and continues with the line following ENDDO.   Both ITERATE and LEAVE are most often used in an IF or IFF command:

```
do while "%var" != "%var1"
  ...
  if "%var" == "%val2" leave
enddo
```

You can nest DO loops up to 15 levels deep.

The DO and ENDDO commands must be on separate lines, and cannot be placed within a command group, or on the same line as other commands (this is the reason DO cannot be used in aliases). However, commands within the DO loop can use command groups or the command separator in the normal way.

You can exit from all DO / ENDDO loops by using GOTO to a line past the last ENDDO. However, be sure to read the cautionary notes about GOTO and DO under the GOTO command before using a GOTO in any other way inside any DO loop.

DRAWBOX

Purpose: Draw a box on the screen.

Format: **DRAWBOX *ulrow ulcol lrrow lrcol style* [BRIght] *fg* ON [BRIght] *bg* [FILl [BRIght] *bgfill*] [ZOOm] [SHAdow]**

ulrow: Row for upper left corner
ulcol: Column for upper left corner
lrrow: Row for lower right corner
lrcol: Column for lower right corner
style: Box drawing style:

- **0** No lines (box is drawn with blanks)
- **1** Single line
- **2** Double line
- **3** Single line on top and bottom, double on sides
- **4** Double line on top and bottom, single on sides

fg: Foreground character color
bg: Background character color
bgfill: Background fill color (for the inside of the box)

See also: DRAWHLINE and DRAWVLINE.

Usage

DRAWBOX is useful for creating attractive screen displays in batch files.

For example, to draw a box around the edge of an 80x25 screen with bright white lines on a blue background:

```
drawbox 0 0 24 79 1 bri whi on blu fill blu
```

See Colors and Color Names for details about colors.

If you use ZOOM, the box appears to grow in steps to its final size. The speed of the zoom operation depends on the speed of your computer and video system.

If you use SHADOW, a drop shadow is created by changing the characters in the row under the box and the 2 columns to the right of the box to normal intensity text with a black background (this will make characters displayed in black disappear entirely).

The row and column values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

DRAWBOX checks for valid row and column values, and displays a "Usage" error message if any values are out of range.

Unlike DRAWHLINE and DRAWVLINE, DRAWBOX does **not** automatically connect boxes to existing lines on the screen with the proper connector characters. If you want to draw lines inside a box and have the proper connectors drawn automatically, draw the box first, then use DRAWHLINE and DRAWVLINE to draw the lines.

DRAWBOX uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or a font which does not include these characters, the box may not appear on your screen as you expect.

DRAWHLINE

Purpose: Draw a horizontal line on the screen.

Format: **DRAWHLINE *row column len style* [BRIght] *fg* ON [BRIght] *bg***

row: Starting row
column: Starting column
len: Length of line
style: Line drawing style:

1	Single line
2	Double line

fg: Foreground character color
bg: Background character color

See also: DRAWBOX and DRAWVLINE.

Usage

DRAWHLINE is useful for creating attractive screen displays in batch files. It detects other lines and boxes on the display, and creates the appropriate connector characters when possible (not all types of lines can be connected with the available characters).

For example, the following command draws a double line along the top row of the display with green characters on a blue background:

```
drawhline 0 0 80 2 green on blue
```

The *row* and *column* values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. DRAWHLINE checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

See Colors and Color Names for details about colors.

DRAWHLINE uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or a font which does not include these characters, the line may not appear on your screen as you expect.

DRAWVLINE

Purpose: Draw a vertical line on the screen.

Format: **DRAWVLINE *row column len style* [BRIght] *fg* ON [BRIght] *bg***

row: Starting row
column: Starting column
len: Length of line
style: Line drawing style:

1	Single line
2	Double line

fg: Foreground character color
bg: Background character color

See also: DRAWBOX and DRAWHLINE.

Usage

DRAWVLINE is useful for creating attractive screen displays in batch files. It detects other lines and boxes on the display, and creates the appropriate connector characters when possible (not all types of lines can be connected with the available characters).

For example, to draw a double width line along the left margin of the display with bright red characters on a black background:

```
drawvline 0 0 25 2 bright red on black
```

The *row* and *column* values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. DRAWVLINE checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

See Colors and Color Names for details about colors.

DRAWVLINE uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or a font which does not include these characters, the line may not appear on your screen as you expect.

ECHO and ECHOERR

Purpose: Display a message, enable or disable batch file or command-line echoing, or display the echo status.

Format: **ECHO [ON | OFF | *message*]**

ECHOERR *message*

message: Text to display.

See also: ECHOS, SCREEN, SCRPUT, SETDOS and TEXT.

Usage

4NT has a separate echo capability for batch files and for the command line. The command-line ECHO state is independent of the batch file ECHO state; changing ECHO in a batch file has no effect on the display at the command prompt, and vice versa.

To see the current echo state, use the ECHO command with no arguments. This displays either the batch file or command-line echo state, depending on where the ECHO command is performed.

In a batch file, if you turn ECHO on, each line of the file is displayed before it is executed. If you turn ECHO off, each line is executed without being displayed. ECHO can also be used in a batch file to display a message on the screen. Regardless of the ECHO state, a batch file line that begins with the [**@**] character will not be displayed. To turn off batch file echoing, without displaying the ECHO command, use this line:

```
@echo off
```

ECHO commands in a batch file will send messages to the screen while the batch file executes, even if ECHO is set OFF. For example, this line will display a message in a batch file:

```
echo Processing your print files...
```

If you want to echo a blank line from within a batch file, enter:

```
echo.
```

You cannot use the command separator character [**&**], or the redirection symbols [**| > <**] in an ECHO message, unless you enclose them in quotes (see Argument Quoting) or precede them with the escape character.

ECHO defaults to ON in batch files. The current ECHO state is inherited by called batch files. You can change the default setting to ECHO OFF with the SETDOS /V0 command, the Options 1 page of the OPTION dialogs, or the BatchEcho directive in the *.INI* file.

If you turn the command-line ECHO on, each command will be displayed before it is executed. This will let you see the command line after expansion of all aliases and variables. The command- line ECHO is most useful when you are learning how to use advanced features. This example will turn command-line echoing on:

```
[c:\] echo on
```

ECHO defaults to OFF at the command line.

ECHOERR acts like ECHO but sends its output to the standard error device (usually the screen) instead of the standard output device. If the standard output of a batch file is redirected to a file or another device with **>**, ECHOERR will still generate a screen message. See Redirection and Piping for more information about the standard output and standard error devices and redirection.

ECHOS and ECHOSERR

Purpose: Display a message without a trailing carriage return and line feed.

Format: **ECHOS message**

ECHOSERR *message*

message: Text to display.

See also: ECHO, SCREEN, SCRPUT, TEXT, and VSCRPUT.

Usage

ECHOS is useful for text output when you don't want to add a carriage return / linefeed pair at the end of the line. For example, you can use ECHOS when you need to redirect control sequences to your printer; this example sends the sequence **Esc P** to the printer on LPT1:

```
[c:\] echos ^eP > lpt1:
```

You cannot use the command separator character [**&**] or the redirection symbols [**| > <**] in an ECHOS message, unless you enclose them in quotes (see Argument Quoting) or precede them with the escape character.

ECHOS does not translate or modify the message text. For example, carriage return characters are not translated to CR/LF pairs. ECHOS sends only the characters you enter (after escape character and back quote processing). The only character you cannot put into an ECHOS message is the NUL character (ASCII 0).

ECHOSERR acts like ECHOS but sends its output to the standard error device (usually the screen) instead of the standard output device. If the standard output of a batch file is redirected to a file or another device with **>**, ECHOSERR will still generate a screen message. See Redirection and Piping for more information about the standard output and standard error devices and redirection.

ENDLOCAL

Purpose: Restore the saved disk drive, directory, environment, alias list, and special characters.

Format: **ENDLOCAL**

See also: SETLOCAL.

Usage

The SETLOCAL command in a batch file saves the current disk drive, default directory, all environment variables, the alias list, and the command separator, escape character, parameter character, decimal separator, and thousands separator. ENDLOCAL restores everything that was saved by the previous SETLOCAL command.

For example, this batch file fragment saves everything, removes all aliases so that user aliases will not affect batch file commands, changes the disk and directory, changes the command separator, runs a program, and then restores the original values:

```
setlocal
unalias *
cdd d:\test
setdos /c~
program ~ echo Done!
endlocal
```

SETLOCAL and ENDLOCAL can only be used in batch files, not in aliases or from the command line.

ESET

Purpose: Edit environment variables and aliases.

Format: **ESET [/A]** ***variable name*****...**

variable name: The name of an environment variable or alias to edit.

/A(lias)

See also: ALIAS, UNALIAS, SET, and UNSET.

Usage

ESET allows you to edit environment variables and aliases using line editing commands (see Command-Line Editing).

For example, to edit the executable file search path:

```
[c:\] eset path
path=c:\;c:\dos;c:\util
```

To create and then edit an alias:

```
[c:\] alias d = dir /d/j/p
[c:\] eset d
d=dir /d/j/p
```

ESET will search for environment variables first and then aliases. If you have an environment variable and an alias with the same name, ESET will edit the environment variable and ignore the alias unless you use the **/A** option.

Environment variable and alias names are normally limited to 80 characters. Their values are normally limited to 1,023 characters. However, if you use special techniques to create a longer environment variable, ESET will edit it provided the variable contains no more than 2,047 characters of text.

If you have enabled global aliases (see ALIAS), any changes made to an alias with ESET will immediately affect all other copies of 4NT which are using the same alias list.

Options

/A: (Alias) Edit the named alias even if an environment variable of the same name exists. If you have an alias and an environment variable with the same name, you must use this switch to be able to edit the alias.

EXCEPT

Purpose: Perform a command on all available files except those specified.

Format: **EXCEPT (*file*) *command***

file: The file or files to exclude from the command.
command: The command to execute, including all appropriate arguments and switches.

See also: ATTRIB and File Exclusion Ranges.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Ranges **must** appear immediately after the EXCEPT keyword.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

EXCEPT provides a means of executing a command on a group of files and/or subdirectories, and excluding a subgroup from the operation. The *command* can be an internal command or alias, an external command, or a batch file.

You may use wildcards to specify the files to exclude from the command. The first example erases all the files in the current directory except those beginning with *MEMO*, and those whose extension is *.WKS*. The second example copies all the files and subdirectories on drive C to drive D except those in *C:\MSC* and *C:\DOS*, using the COPY command:

```
[c:\] except (memo*.* *.wks) erase *.*
[c:\] except (c:\msc c:\dos) copy c:\*.* d:\ /s
```

When you use EXCEPT on an HPFS, NTFS, or LFN drive, you must quote any file names inside the parentheses which contain whitespace or special characters. See File Names for additional details. For example, to copy all files except those in the *"Program Files"* directory to drive E:\:

```
[c:\] except ("Program Files") copy /s *.* e:\
```

EXCEPT prevents operations on the specified file(s) by setting the hidden attribute, performing the command, and then clearing the hidden attribute. If the command is aborted in an unusual way, you may need to use the ATTRIB command to remove the hidden attribute from the file(s).

Caution: EXCEPT will not work with programs or commands that ignore the hidden attribute or which work explicitly with hidden files, including DEL /Z, and the **/H** (process hidden files) switch available in some 4NT file processing commands.

File exclusion ranges provide a faster and more flexible method of excluding files from internal commands, and do not manipulate file attributes, as EXCEPT does. However, exclusion ranges can only be used with 4NT internal commands; you must use EXCEPT for external commands.

Date, time, and size ranges can be used immediately after the word EXCEPT to further qualify which files should be excluded from the *command*. If the *command* is an internal command that supports ranges, an independent range can also be used in the *command* itself. You can also use a file exclusion range within the EXCEPT command; however, this will select files to be **excluded** from EXCEPT, and therefore **included** in execution of the *command*.

You can use command grouping to execute multiple *commands* with a single EXCEPT.   For example, the following command copies all files in the current directory whose extensions begin with *.DA*, except the *.DAT* files, to the *D:\SAVE* directory, then changes the first two characters of the extension of the copied files to *.SA*:

```
[c:\data] except (*.dat) (copy *.da* d:\save & ren *.da* *.sa*)
```

If you use filename completion to enter the filenames inside the parentheses, type a space after the open parenthesis before entering a partial filename or pressing **Tab**.   Otherwise, the command-line editor will treat the open parenthesis as the first character of the filename to be completed.

EXIT

Purpose: Return from 4NT.

Format: **EXIT [*value*]**

value: The numeric exit code to return.

Usage

EXIT terminates the current copy of 4NT. Use it to return to an application when you have "shelled out" to work at the prompt, or to end an Windows NT command-line session.

To close the session, or to return to the application that started 4NT, type:

```
[c:\] exit
```

If you specify a *value*, EXIT will return that value to the program that started 4NT. For example:

```
[c:\] exit 255
```

The *value* is a number you can use to inform the program of some result, such as the success or failure of a batch file. It can range from 0 - 4,294,967,295. This feature is most useful for systems which use batch files to automate their operation, such as bulletin boards, or custom application programs like databases that shell to 4NT to perform certain tasks.

FFIND

Purpose: Search for files by name or contents.

Format: **FFIND [/A[[:][-]rhsda] /B /C /D[*list*] /E /I /K /L /M /O[[:][-]acdeginrsu] /P /R /S /T"xx" /V /X["xx xx ..."]] file...**

list: A list of disk drive letters (without colons).
file: The file, directory, or list of files or directories to display.

/A(ttribute select)	**/M** (no footers)
/B(are)	**/O**(rder)
/C(ase sensitive)	**/P**(ause)
/D(rive)	**/R**(everse)
/E (upper case display)	**/S**(ubdirectories)
/I(gnore wildcards)	**/T"xx"** (Text search string)
/K (no headers)	**/V**(erbose)
/L(ine numbers)	**/X** (hex display / search string)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

FFIND is a flexible search command that looks for files based on their names and their contents. Depending on the options you choose, FFIND can display filenames, matching text, or a combination of both in a variety of formats.

If you want to search for files by name, FFIND works much like the DIR command. For example, to generate a list of all the *.BTM* files in the current directory, you could use the command

```
[c:\] ffind *.btm
```

The output from this command is a list of full pathnames, followed by the number of files found.

If you want to limit the output to a list of *.BTM* files which contain the string *color*, you could use this command instead:

```
[c:\] ffind /t"color" *.btm
```

The output from this command is a list of files that contain the string *color* along with the first line in each file that contains that string. By default, FFIND uses a case-insensitve search, so the command above will include files that contain *COLOR*, *Color*, *color*, or any other combination of upper-case and lower-case letters.

If you would rather see the **last** line of each file that contains the search string, use the **/R** option, which forces FFIND to search from the end of each file to the beginning. This option will also speed up searches somewhat if you are looking for text that will normally be at the end of a file, such as a signature line:

```
[c:\] ffind /r /t"Sincerely," *.txt
```

You can use extended wildcards in the search string to increase the flexibility of FFIND's search. For example, the following command will find *.TXT* files which contain either the string *June* or *July* (it will also find *Juny* and *Jule*). The **/C** option makes the search case-sensitive:

```
[c:\] ffind /c /t"Ju[nl][ey]" *.txt
```

If you want to search for text that contains wildcard characters (*****, **?**, **[**, or **]**), you can use the **/I** option to force FFIND to interpret these as normal characters instead of wildcards. The following command, for example, finds all *.TXT* files that contain a question mark:

```
[c:\] ffind /i /t"?" *.txt
```

At times, you may need to search for data that cannot be represented by ASCII characters. You can use FFIND's **/X** option to represent the search string in hexadecimal format (this option also changes the output to show hexadecimal offsets rather than text lines). With **/X**, the search must be represented by pairs of hexadecimal digits separated by spaces; a search of this type is always case-sensitive (in the example below, 41 63 65 is the hex code for "Ace"):

```
[c:\] ffind /x"41 63 65" *.txt
```

You can use FFIND's other options to further specify the files for which you are searching and to modify the way in which the output is displayed.

When you use FFIND on an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names for additional details.

Options

/A (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **FFIND /A ...**), FFIND will search all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included in the listing. For example, **/A:RHS** will search only those files with all three attributes set.

/B (Bare) Display file names only and omit the text that matches the search. This option is only useful in combination with **/T** or **/X**, which normally force FFIND to display file names and matching text.

/C (Case sensitive) Perform a case-sensitive search. This option is only valid with **/T**, which defaults to a case-insensitive search. It is not needed with a **/X** hexadecimal search, which is always case-sensitive.

/D (Drive) Search all files on one or more drives. If you use **/D** without a list of drives, FFIND will search the drives specified in the list of files. If no drive letters are listed, FFIND will search all of the current drive. You can include a list of drives or a range of drives to search as part of the **/D** option. For example, to search drives C:, D:, E:, and G:, you can use either of these commands:

```
[c:\] ffind /dcdeg ...
```

```
[c:\] ffind /dc-eg ...
```

Drive letters listed after **/D** will be ignored when processing file names which also include a drive letter. For example, this command displays all the *.BTM* files on C: and E:, but only the *.BAT* files on D:

```
[c:\] ffind /s /dce *.btm d:\*.bat
```

/E Display filenames in the traditional upper case; also see SETDOS /U and the UpperCase directive in *4NT.INI*.

/I (Ignore wildcards) Only meaningful when used in conjunction with the **/T "text"** option. Suppresses the recognition of wildcard characters in the search text. This option is useful if you need to search for characters that would normally be interpreted as wildcards: *****, **?**, **[**, and **]**.

/K (No headers) Suppress the display of the header or filename for each matching text line.

/L (Line numbers) Include the line number for each text line displayed.

/M (No footers) Suppress the footer (the number of files and number of matches) at the end of FFIND's display.

/O (Sort order) Set the sort order for the files that FFIND displays. You may use any combination of the following sorting options; if multiple options are used, the listing will be sorted with the first sort option as the primary key, the next as the secondary key, and so on:

-	Reverse the sort order for the next option
a	Sort in ASCII order, not numerically, when there are digits in the name
d	Sort by date and time (oldest first); for drives which support long file names.
e	Sort by extension
g	Group subdirectories first, then files
i	Sort by file description
n	Sort by filename (this is the default)
r	Reverse the sort order for all options
s	Sort by size
u	Unsorted

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/R (Reverse) Only meaningful when used in conjuction with the **/T "text"** or **/X** options. Searches each file from the end backwards to the beginning. This option is useful if you want to display the last occurrence of the search string in each file instead of the first (the default). It can also speed up searches for information that is normally at the end of a file, such as a signature.

/S (Subdirectories) Display matches from the current directory and all of its subdirectories.

/T"xx" (Text search) Specify the text search string. **/T** must be followed by a text string in double quotes (*e.g.*, **/t"color"**). FFIND will perform a case-insensitive search unless you also use the **/C** option. For a hexadecimal search and/or hexadecimal display of the location where the search string is found, see **/X**. You can specify a search string with either **/T** or **/X**, but not both.

/V (Verbose) Show every matching line. FFIND's default behavior is to show only the first matching line then and then go on to the next file. This option is only valid with **/T** or **/X**.

/X["xx xx ..."] (Hexadecimal display / search) Specify hexadecimal display and an optional hexadecimal search string.

If **/X** is followed by one or more pairs of hexadecimal digits in quotes (*e.g.*, **/x"44 63 65"**), FFIND will search for that exact sequence of characters or data bytes without regard to the meaning of those bytes as text. If those bytes are found, the offset is displayed (in both decimal and hexadecimal). A search of this type will always be case-sensitive.

If **/X** is **not** followed by a hexadecimal search string it must be used in conjunction with **/T**, and will change the output format to display offsets (in both decimal and hexadecimal) rather than actual text lines when the search string is found. For example, this command uses **/T** to display the first line in each *.BTM* file containing the word "hello":

```
[c:\] ffind /t"hello" *.btm
---- c:\test.btm
echo hello

1 line in 1 file
```

If you use the same command with **/X**, the offset is displayed instead of the text:

```
[c:\] ffind /t"hello" /x *.btm
---- c:\test.btm
Offset: 26 (1Ah)

1 line in 1 file
```

You can specify a search string with either **/T** or **/X**, but not both.

FOR

Purpose: Repeat a command for several values of a variable.

Format: **FOR [/A:[[-]rhsda] /F ["*options*"] /H /L /R [*path*]] *%var* IN ([@]*set* | *start, step, end*) [DO] *command* ...**

options: Parsing options for a "file parsing" **FOR**.
path: The starting directory for a "recursive" **FOR**.
%var: The variable to be used in the command ("FOR variable").
set: A set of values for the variable.
start: The starting value for a "counted" **FOR**.
step: The increment value for a "counted" **FOR**.
end: The limit value for a "counted" **FOR**.
command: A command or group of commands to be executed for each value of the variable.

/A: (Attribute select)	**/L** (counted loop)
/F(ile parsing)	**/R**(ecursive)
/H(ide dots)	

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Ranges **must** appear immediately after the FOR keyword.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

FOR begins by creating a set. It then executes a command for every member of the set. The command can be an internal command, an alias, an external command, or a batch file. The members of the set can be a list of file names, text strings, a group of numeric values, or text read from a list of files.

When the *set* is made up of text or several separate file names (not an include list), the elements must be separated by spaces, tabs, commas, or the switch character (normally a slash [**/**]).

FOR includes a large number of options, some of which duplicate functions available in other 4NT commands, and / or do not follow conventions you may find in our other commands. Most of these extra options are included for compatibility with Windows NT 4.0's *CMD.EXE*.

The first three sections below (***Working with Files***, ***Working with Text***, and ***Retrieving Text from Files***) describe the traditional FOR command and the enhancements to it which are part of 4NT. The sections on ***Parsing Text from Files*** and ***Counted FOR Loops*** describe features added for compatibility with Windows NT 4.0. The section entitled ***Other Notes*** contains information you may need if you use any aspect of the FOR command extensively.

Working with Files

Normally, the *set* is a list of files specified with wildcards. For example, if you use this line in a batch file:

```
for %x in (*.txt) do list %x
```

then LIST will be executed once for each file in the current directory with the extension *.TXT*. The FOR variable %x is set equal to each of the file names in turn, then the LIST command is executed for each file.   (You could do the same thing more easily with a simple LIST *.TXT.   We used FOR here so you could get a feel for how it operates, using a simple example.   Many of the examples in this section are constructed in the same way.)

The *set* can include multiple files or an include list, like this:

```
        for %x in (d:\*.txt;*.doc;*.asc) do type %x
```

FOR supports wildcards and extended wildcards, as well as extended parent directory names (*e.g.*, **...*.txt** to process all of the *.TXT* files that are contained in the directory 2 levels above the current directory).

When you use FOR on an HPFS, NTFS, or LFN drive, you must quote any file names within the *set* which contain whitespace or special characters.   The same restriction applies to names returned in the FOR variable, if you pass them to 4NT internal commands, or other commands which require quoting filenames with whitespace.   FOR does not quote returned names automatically, even if you included quotes in the *set*.   See File Names for additional details on file name quoting.

If the *set* includes filenames, the file list can be further refined by using date, time, size, and file exclusion ranges.   The range or ranges must be placed immediately after the word FOR.   Ranges will be ignored if no wildcards are used inside the parentheses.   For example, this *set* is made up of all of the *.TXT* files that were created or updated on October 4, 1997:

```
        for /[d10-4-97,+0] %x in (*.txt) do ...
```

If the *command* is an internal command that supports ranges, an independent range can also be used in the *command* itself.

You can also refine the list by limiting it with the **/A** option to select only files that have specific attributes.

By default, FOR works only with files in the current directory or a specified directory.   With the **/R** option, FOR will also search for files in subdirectories.   For example, to work with all of the *.TXT* files in the current directory and its subdirectories:

```
        for /r %x in (*.txt) do ...
```

If you specify a directory name immediately after **/R**, FOR will start in that directory and then search each of its subdirectories.   This example works with all of the *.BAK* files on drive D:

```
        for /r d:\ %x in (*.bak) do ...
```

When you use wildcards to specify the *set*, FOR scans the directory and finds each file which matches the wildcard name(s) you specified.   If, during the processing of the FOR command, you create a file that could be included in the *set*, it may or may not appear in a future interation of the same FOR command.   Whether the new file appears depends on its physical location in the directory structure.   For example, if you use FOR to execute a command for all *.TXT* files, and the command also creates one or more new *.TXT* files, those new files may or may not be processed during the current FOR command, depending on where they are placed in the physical structure of the directory.   This is an operating system constraint over which 4NT has no control.   Therefore, in order to achieve consistent results you should construct FOR commands which do not create files that could become part of the *set* for the current command.

Working with Text

The set can also be made up of text instead of file names.   For example, to create three files named file1, file2, and file3, each containing a blank line:

```
        for %suffix in (1 2 3) do echo. > file%suffix
```

You could also use the names of environment variables as the text. This example displays the name and content of several variables from the environment (see Environment Variables and Functions for details on the use of square brackets when expanding environment variables). Enter this on one line:

```
        for %var in (path prompt comspec) do echo %var=%[%var]
```

Retrieving Text from Files

FOR can extract text from files in two different ways. The first method extracts each line from each file in the set and places it in the variable. To use this method, place an [**@**] at the beginning of the set, in front of the file name or names.

For example, if you have a file called *DRIVES.TXT* that contains a list of drives on your computer, one drive name per line (with a ":" after each drive letter), you can print the free space on each drive this way:

```
        for %d in (@drives.txt) do free %d > prn
```

Because the [**@**] is also a valid filename character, FOR first checks to see if the file exists with the [**@**] in its name (*i.e.*, a file named @*DRIVES.TXT*). If so, the filename is treated as a normal argument. If it doesn't exist, FOR uses the filename (without the [**@**]) as the file from which to retrieve text.

If you use @CON as the filename, FOR will read from standard input (a redirected input file) or a pipe (see Redirection and Piping). If you use @CLIP: as the filename, FOR will read any text available from the Windows clipboard.

Parsing Text from Files

The second method of working with text from files is to have FOR parse each line of each file for you. To begin a "file-parsing" FOR, you must use the **/F** option and then include one or more file names in the *set*. When you use this form of FOR, the variable must be a single letter, for example, *%a*.

This method of parsing, included for compatibility with Windows NT 4.0's *CMD.EXE*, can be cumbersome and inflexible. For a more powerful method, use FOR with **@filename** as the ***set*** to retrieve each line from the file, as described in the previous section. Then use variable functions like @INSTR, @LEFT, @RIGHT, and @WORD to parse the line.

By default, FOR will extract the first word or ***token*** from each line and return it in the variable. For example, to display the first word on each line in the file *FLIST.TXT*:

```
        for /f %a in (flist.txt) do echo %a
```

You can control the way FOR /F parses each line by specifying one or more parsing options in a quoted string immediately after the **/F**. The available options are:

skip=n: FOR /F will skip "n" lines at the beginning of each file before parsing the remainder of the file.

tokens=n, m, ... : By default, FOR /F returns just the first word or "token" from each parsed line in the variable you named. You can have it return more than one token in the variable, or return tokens in several variables, with this option.

This option is followed by a list of numbers separated by commas. The first number tells FOR /F which token to return in the first variable, the second number tells it which to return in the second variable, etc. The variables follow each other alphabetically starting with the variable you name on the FOR command line.   This example returns the first word of each line in each text file in *%d*, the second in *%e*, and the third in *%f*:

```
for /f "tokens=1,2,3" %d in (*.txt) do ...
```

You can also indicate a range of tokens by separating the numbers with a hyphen [**-**].   This example returns the first word of each line in %p, the second through fifth words in %q, and the eighth word in %r:

```
for /f "tokens=1,2-5,8" %p in (*.txt) do ...
```

To return the rest of the line in a variable, use a range that ends with a number higher than the last token in any line, such as 999.   This final example returns the first word of each line in *%a* and the remainder of each line (assuming that no line has more than 999 words!) in *%b*:

```
for /f "tokens=1,2-999" %a in (*.txt) do ...
```

eol=c: If FOR /F finds the character "c" in the line, it will assume that the character and any text following it are part of a comment and ignore the rest of the line.

delims=xxx...:   By default, FOR /F sees spaces and tabs as word or token delimiters.   This option replaces those delimiters with all of the characters following the equal sign to the end of the string.   This option must therefore be the last one used in the quoted options string.

You can also use FOR /F to parse a single string instead of each line of a file by using the string, in quotes, as the *set*.   For example, this command will assign variable A to the string "this", B to "is", etc., then display "this" (enter the command on one line):

```
for /f "tokens=1,2,3,4" %a in ("this is a test") do echo %a
```

"Counted" FOR Loop

The "counted FOR" loop is included only for compatibility with Windows NT 4.0's *CMD.EXE*.   In most cases, you will find the DO command more useful for performing counted loops.

In a counted FOR command, the *set* is made up of numeric values instead of text or file names.   To begin a counted FOR command, you must use the **/L** option and then include three values, separated by commas, in the *set*.   These are the *start*, *step*, and *end* values.   During the first iteration of the FOR loop, the variable is set equal to the *start* value.   Before each iteration, the variable is increased by the *step* value.   The loop ends when the variable exceeds the *end* value.   This example will print the numbers from 1 to 10:

```
for /l %val in (1,1,10) do echo %val
```

This example will print the odd numbers from 1 to 10 (1, 3, 5, 7, and 9):

```
for /l %val in (1,2,10) do echo %val
```

The *step* value can be negative.   If it is, the loop will end when the variable is less than the *end* value.

Other Notes

You can use either *%* or *%%* in front of the variable name. Either form will work, whether the FOR command is typed from the command line or is part of an alias or batch file (some of the traditional command processors require a single *%* if FOR is used at the command line, but require *%%* if FOR is used in a batch file).   The variable name can be up to 80 characters long.   The word DO is optional.

If you use a single-character FOR variable name, that name is given priority over any environment variable which starts with the same letter, in order to maintain compatibility with the traditional FOR command.   For example, the following command tries to add a: and b: to the end of the PATH, but will not work as intended:

```
[c:\] for %p in (a: b:) do path %path;%p
```

The "%p" in "%path" will be interpreted as the FOR variable %p followed by the text "ath", which is not what was intended.   To get around this, use a different letter or a longer name for the FOR variable, or use square brackets around the variable name (see Environment).

The following example uses FOR with variable functions to delete the *.BAK* files for which a corresponding *.TXT* file exists in the current directory:

```
[c:\docs] for %file in (*.txt) do del %@name[%file].bak
```

The above command would not work properly on an HPFS, NTFS, or LFN drive, because the returned FILE variable might contain whitespace.   To correct this problem, you would need two sets of quotes, one for DEL and one for %@NAME:

```
[c:\docs] for %file in (*.txt) do del "%@name["%file"].bak"
```

You can use command grouping to execute multiple commands for each element in the *set*.   For example, the following command copies each *.WKQ* file in the current directory to the *D:\WKSAVE* directory, then changes the extension of each file in the current directory to *.SAV*.   This should be entered on one line:

```
[c:\text] for %file in (*.wkq) do (copy "%file" d:\wksave\ & ren "%file"
           *.sav)
```

In a batch file you can use GOSUB to execute a subroutine for every element in the *set*.   Within the subroutine, the FOR variable can be used just like any environment variable.   This is a convenient way to execute a complex sequence of commands for every element in the *set* without CALLing another batch file.

One unusual use of FOR is to execute a collection of batch files or other commands with the same parameter.   For example, you might want to have three batch files all operate on the same data file.   The FOR command could look like this (enter on one line):

```
[c:\] for %cmd in (filetest fileform fileprnt) do %cmd datafile
```

This line will expand to three separate commands:

```
filetest datafile
fileform datafile
fileprnt datafile
```

The variable that FOR uses (the **%CMD** in the example above) is created in the environment and then erased when the FOR command is done.   (For compatibility with *CMD.EXE*, a single-character FOR variable is created in a special way that does not overwrite an existing environment variable with the same name.)   When using a multi-character variable name you must be careful not to use the name of one of your environment variables as a FOR variable.   For example, a command that begins

```
[c:\] for %path in ...
```

will write over your current path setting and then erase the path variable completely when FOR is done.

FOR statements can be nested.

Options

/A: (Attribute select) Process only those files that have the specified attribute(s). **/A** will be used only when processing wildcard file names in the *set*. It will be ignored for filenames without wildcards or other items in the *set*. Preceding the attribute character with a hyphen [**-**] will process files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed (*e.g.*, **FOR /A: ...**), FOR will process all files including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included. For example, **/A:RHS** will include only those files with all three attributes set.

For example, to process only those files with the archive attribute set:

```
for /a:a %f in (*.*) echo %f needs a backup!
```

/F (File parsing) Return one or more words or tokens from each line of each file in the set. The **/F** option can be followed by one or more options in a quoted string which control how the parsing is performed. See the details under **Parsing Text From Files**, above.

/H (Hide dots) Suppress the assignment of the "." and ".." directories to the FOR variable.

/L (counted loop) Interpret the three values in the *set* as the start, step, and end values of a counted loop. See the details under **"Counted" FOR Loop**, above.

/R (Recursive) Look in the current directory and all of its subdirectories for files in the *set*. If the **/R** is followed by a directory name, look for files in that directory and all of its subdirectories.

FREE

Purpose: Display the total disk space, total bytes used, and total bytes free on the specified (or default) drive(s).

Format: **FREE [*drive:* ...]**

drive: One or more drives to include in the report.

See also: MEMORY.

Usage

FREE provides the same disk information as the external command CHKDSK, but without the wait, since it does not check the integrity of the file and directory structure of the disk.

A colon [**:**] is required after each drive letter. This example displays the status of drives A and C:

```
[c:\] free a: c:
 Volume in drive A: is unlabeled
  1,213,952 bytes total disk space
  1,115,136 bytes used
     98,816 bytes free
 Volume in drive C: is DEVELOPMENT
 242,496,000 bytes total disk space
 236,851,712 bytes used
   5,644,288 bytes free
```

FTYPE

Purpose: Modify or display the command used to open a file of a type specified in the Windows NT registry.

Format: **FTYPE [/P] [*filetype*[=[*command*]]]**

filetype: A file type stored in the Windows NT registry.
command: The command to be executed when a file of the specified type is opened.

/P(ause)

See also: ASSOC and Executable Extensions.

Usage

FTYPE allows you to display or update the command used to open a file of a specified type listed in the Windows NT registry.

FTYPE modifies the behavior of "indirect" Windows file associations stored under the registry handle HKEY_CLASSES_ROOT, and discussed in more detail Windows File Associations and Using Windows File Associations. If you are not familiar with file associations be sure to read about them before using FTYPE.

The entry modified by FTYPE is the **Shell\Open\Command** entry for the specified file type, which defines the application to execute when a file of that type is opened. The open action is generally invoked by selecting **Open** on the popup menu for a file from the Windows NT Explorer. Note that opening a file and double-clicking its icon (or selecting the icon and pressing Enter) may not be the same thing — double-clicking or pressing Enter invokes the default action for the file type, which may or may not be "Open".

If you invoke FTYPE with no parameters, it will display the current file types and associated shell open commands. Use the **/P** switch to pause the display at the end of each page. If you include a *filetype*, with no equal sign or *command*, FTYPE will display the current command for that file type.

If you include the equal sign and *command*, FTYPE will create or update the shell open command for the specified file type. The *command* generally includes an application name, including full path, plus parameters. The specific syntax required depends on the internal operation of both Windows and the application involved, and is beyond the scope of this manual. You can learn about typical syntax by reviewing appropriate Windows and application documentation, and / or by checking through the current contents of your registry.

To remove the shell open command for a file type, use a command like **FTYPE filetype=**, with no *command* parameter. This will not delete the shell open command entry from the registry; it simply sets the command to an empty string.

FTYPE should be used with caution, and only after backing up the registry. Improper changes to file associations can prevent applications and / or the operating system from working properly.

Options

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

GLOBAL

Purpose: Execute a command in the current directory and its subdirectories.

Format: **GLOBAL [/H /I /P /Q]** ***command***

command: The command to execute, including arguments and switches.

/H(idden directories)	**/P**(rompt)
/I(gnore exit codes)	**/Q**(uiet)

Usage

GLOBAL performs the command first in the current directory and then in every subdirectory under the current directory. The command can be an internal command, an alias, an external command, or a batch file.

This example copies the files in every directory on drive A to the directory *C:\TEMP*:

```
[a:\] global copy *.* c:\temp
```

If you use the **/P** option, GLOBAL will prompt for each subdirectory before performing the command. You can use this option if you want to perform the command in most, but not all subdirectories of the current directory.

You can use command grouping to execute multiple *commands* in each subdirectory. For example, the following command copies each *.TXT* file in the current directory and all of its subdirectories to drive A. It then changes the extension of each of the copied files to *.SAV*:

```
[c:\] global (copy *.txt a: & ren *.txt *.sav)
```

Options

/H (Hidden directories) Forces GLOBAL to look for hidden directories. If you don't use this switch, hidden directories are ignored.

/I (Ignore exit codes) If this option is not specified, GLOBAL will terminate if the command returns a non- zero exit code. Use **/I** if you want the command to continue in additional subdirectories even if it returns an error in one subdirectory. Even if you use **/I**, GLOBAL will normally halt execution if the command processor receives a **Ctrl-C** or **Ctrl-Break**.

/P (Prompt) Forces GLOBAL to prompt with each directory name before it performs the command. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Do not display the directory names as each directory is processed.

GOSUB

Purpose: Execute a subroutine in the current batch file.

Format: **GOSUB *label***

label: The batch file label at the beginning of the subroutine.

See also: CALL, GOTO and RETURN.

Usage

GOSUB can only be used in batch files.

4NT allows subroutines in batch files. A subroutine must start with a *label* (a colon [**:**] followed by a one-word label name) which appears on a line by itself. Case differences are ignored when matching labels. Labels may be one or more words long. The subroutine must end with a RETURN statement.

The subroutine is invoked with a GOSUB command from another part of the batch file. After the RETURN, processing will continue with the command following the GOSUB command. For example, the following batch file fragment calls a subroutine which displays the directory and returns:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit

:subr1
dir /a/w
return
```

4NT begins its search on the line of the batch file following the GOSUB command. If the *label* is not found between the current position and the end of the file, GOSUB will restart the search at the beginning of the file. If the *label* does not exist, the batch file is terminated with the error message "Label not found."

GOSUB saves the IFF and DO states, so IFF and DO statements inside a subroutine won't interfere with IFF statements in the part of the batch file from which the subroutine was called.

Subroutines can be nested.

GOTO

Purpose: Branch to a specified line inside the current batch file.

Format: **GOTO [/I]** ***label***

label: The batch file label to branch to.

/I(FF and DO continue)

See also: GOSUB.

Usage

GOTO can only be used in batch files.

After a GOTO command in a batch file, the next line to be executed will be the one immediately after the *label*. The *label* must begin with a colon [**:**] and appear on a line by itself. The colon is required on the line where the label is defined, but is not required in the GOTO command itself. Case differences are ignored when matching labels. Labels may be one or more words long.

This batch file fragment checks for the existence of the file *CONFIG.NT*. If the file exists, the batch file jumps to C_EXISTS and copies all the files from the current directory to the root directory on A:. Otherwise, it prints an error message and exits.

```
if exist config.nt goto C_EXISTS
echo CONFIG.NT doesn't exist - exiting.
quit
:C_EXISTS
copy *.* a:\
```

4NT begins its search on the line of the batch file following the GOTO command. If the *label* is not found between the current position and the end of the file, GOTO will restart the search at the beginning of the file. If the *label* does not exist, the batch file is terminated with the error message "Label not found."

To avoid errors in the processing of nested statements and loops, GOTO cancels all active IFF statements and DO / ENDDO loops unless you use **/I**. This means that a normal GOTO (without **/I**) may not branch to any label that is between an IFF and the corresponding ENDIFF or between a DO and the corresponding ENDDO.

For compatibility with Windows NT's *CMD.EXE*, the command

```
GOTO :EOF
```

will end processing of the current batch file if the label :EOF does not exist. However, this is less efficient than using the QUIT or CANCEL command to end a batch file.

Options

/I (IFF and DO continue) Prevents GOTO from canceling IFF statements and DO loops. Use this option only if you are absolutely certain that your GOTO command is branching entirely within any current IFF statement **and** any active DO / ENDDO block. Using **/I** under any other conditions will cause an error later in your batch file.

You cannot branch into another IFF statement, another DO loop, or a different IFF or DO nesting level, whether you use the **/I** option or not.   If you do, you will eventually receive an "unknown command" error (or execution of the UNKNOWN_CMD alias) on a subsequent ENDDO, ELSE, ELSEIFF, or ENDIFF statement.

HELP

Purpose: Display help for internal commands, and optionally for external commands.

Format: **HELP [*topic*]**

topic: A help topic, internal command, or external command.

Usage

Online help is available for 4NT. The 4NT help system uses the Windows NT help facility.

If you type the command HELP by itself (or press **F1** when the command line is empty), the table of contents is displayed. If you type HELP plus a topic name, that topic is displayed. For example:

```
help copy
```

displays information about the COPY command and its options.

HISTORY

Purpose: Display, add to, clear, or read the history list.

Format: **HISTORY [/A *command* /F /P /R *filename***

command: A command to be added to the history list.
filename: The name of a file containing entries to be added to the history list.

/A(dd)	**/P**(ause)
/F(ree)	**/R**(ead)

See also: DIRHISTORY and LOG.

Usage

4NT keeps a list of the commands you have entered on the command line. See Command History and Recall for additional details.

The HISTORY command lets you view and manipulate the command history list directly. If no parameters are entered, HISTORY will display the current command history list:

```
[c:\] history
```

With the options explained below, you can clear the list, add new commands to the list without executing them, save the list in a file, or read a new list from a file.

The number of commands saved in the history list depends on the length of each command line. The history list size can be specified at startup from 256 to 32767 characters (see the History directive). The default size is 1024 characters.

Your history list can be stored either locally (a separate history list for each copy of 4NT) or globally (all copies of 4NT share the same list). For full details see the discussion of local and global history lists under Command History and Recall.

You can use the HISTORY command as an aid in writing batch files by redirecting the HISTORY output to a file and then editing the file appropriately. However, it may be easier to use the LOG /H command for this purpose.

You can disable the history list or specify a minimum command-line length to save on the Command Line 1 page of the OPTION dialogs, or with the HistMin directive in the *.INI* file.

You can save the history list by redirecting the output of HISTORY to a file. This example saves the command history to a file called *HISTFILE* and reads it back again immediately. If you leave out the HISTORY /F command on the second line, the contents of the file will be appended to the current history list instead of replacing it:

```
[c:\] history > histfile
[c:\] history /f
[c:\] history /r histfile
```

If you need to save your history at the end of each day's work, you might use commands like this in your *4START.BTM* file:

```
if exist c:\histfile history /r c:\histfile
```

```
    alias shut*down `history > c:\histfile`
```

This restores the previous history list if it exists, then defines an alias which will allow you to save the history before shutting off the system.

Options

/A (Add) Add a command to the history list. This performs the same function as the **Ctrl-K** key at the command line (see Command History and Recall).

/F (Free) Erase all entries in the command history list.

/P (Prompt) Wait for a key after displaying each page of the list. Your options at the prompt are explained in detail under Page and File Prompts.

/R (Read) Read the command history from the specified file and append it to the history list currently held in memory. Each line in the file must fit within the command-line length limit).

If you are creating a HISTORY /R file by hand, and need to create an entry that spans multiple lines in the file, you can do so by terminating each line, except the last, with an escape character. However, you cannot use this method to exceed the command-line length limit.

IF

Purpose: Execute a command if a condition or set of conditions is true.

Format: **IF [/I] [NOT] *condition* [.AND. | .OR. | .XOR. [NOT] *condition* ...] *command***

condition: A test or set of tests to determine if the command should be executed.
command: The command to execute if the condition is true.

/I(gnore case)

See also: IFF, @IF

Usage

IF is normally used only in aliases and batch files. It is always followed by one or more *conditions* and then a *command*. First, the *conditions* are evaluated. If they are true, the *command* is executed. Otherwise, the *command* is ignored. If you add a NOT before a *condition*, the *command* is executed only when the *condition* is false.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can group conditions with parentheses (see **Combining Tests** below). You can also nest IF statements.

The *conditions* can test strings, numbers, the existence of a file or subdirectory, the exit code returned by the preceding external command, and the existence of aliases and internal commands.

The *command* can be an alias, an internal command, an external command, or a batch file. The entire IF statement, including all *conditions* and the *command*, must fit on one line.

Some examples of IF conditions and commands are included below; additional examples can be found in the *EXAMPLES.BTM* file which came with 4NT.

You can use command grouping to execute multiple *commands* if the *condition* is true. For example, the following command tests if any *.TXT* files exist. If they do, they are copied to drive A: and their extensions are changed to *.TXO*:

```
if exist *.txt (copy *.txt a: & ren *.txt *.txo)
```

(Note that the IFF command provides a more structured method of executing multiple commands if a condition or set of conditions is true.)

When an IF test fails, the remainder of the command is discarded, and 4NT normally continues with the next command on the line, or the next line. This behavior is not compatible with *CMD.EXE*, which discards all remaining commands on the line when an IF test fails, including those after a command separator or pipe character. To change the behavior so that IF affects all commands on the line, as in *CMD.EXE*, set DuplicateBugs to Yes in the *.INI* file.

For example, if DuplicateBugs is set to Yes (the default), the following command will display nothing, because the second ECHO command is discarded along with the first when the ***condition*** fails. If DuplicateBugs is set to No, it will display "hello":

```
[c:\] if 1 == 2 echo Wrong! & echo hello
```

Conditions

The conditional tests listed in the following sections are available in both the IF and IFF commands. They fit into two categories: string and numeric tests, and status tests. The tests can use environment variables, internal variables and variable functions, file names, literal text, and numeric values as their arguments.

String and Numeric Tests

Six test conditions can be used to test character strings. The same conditions are available for both numeric and normal text strings (see below for details). In each case you enter the test as:

```
string1 operator string2
```

The **operator** defines the type of test (equal, greater than or equal, and so on). You should always use spaces on both sides of the **operator**. The operators are:

EQ or **==**	*string1* equal to *string2*
NE or **!=**	*string1* not equal to *string2*
LT	*string1* less than *string2*
LE	*string1* less than or equal to *string2*
GE	*string1* greater than or equal to *string2*
GT	*string1* greater than *string2*

When IF compares two character strings, it will use either a **numeric** comparison or a **string** comparison. A numeric comparison treats the strings as numeric values and tests them arithmetically. A string comparison treats the strings as text.

The difference between numeric and string comparisons is best explained by looking at the way two values are tested. For example, consider comparing the values 2 and 19. Numerically, 2 is smaller, but as a string it is "larger" because its first digit is larger than the first digit of 19. So the first of these *condition* s will be true, and the second will be false:

```
if 2 lt 19 ...
if "2" lt "19" ...
```

IF determines which kind of test to do by examining the first character of each string. If both strings begin with a numeric character (a digit, sign, or decimal separator), a numeric comparison is used. (If a string begins with a decimal separator it is not considered numeric unless the next character is a digit, and there are no more decimal separators within the string. For example, ".07" is numeric, but ".a" and ".07.01" are not.) If either value is non-numeric, a string comparison is used. To force a string comparison when both values are or may be numeric, use double quotes around the values you are testing, as shown above. Because the double quote is not a numeric character, IF performs a string comparison.

Case differences are ignored in string comparisons. If two strings begin with the same text but one is shorter, the shorter string is considered to be "less than" the longer one. For example, "a" is less than "abc", and "hello_there" is greater than "hello".

When you compare text strings, you should always enclose the arguments in double quotes in order to avoid syntax errors which may occur if one of the argument values is empty.

Numeric comparisons work with both integer and decimal values. The values to be compared must contain only numeric digits, decimal points, and an optional sign (**+** or **-**). The number may contain up to 16 digits to the left of the decimal point, and 8 digits to the right.

In order to maintain compatibility with *CMD.EXE* in Windows NT 4.0 and later, 4NT recognizes the following additional names for conditions:

EQL	is the same as **EQ** and **==**
NEQ	is the same as **NE** and **!=**
LSS	is the same as **LT**
LEQ	is the same as **LE**
GTR	is the same as **GT**
GEQ	is the same as **GE**

Internal variables and variable functions are very powerful when combined with string and numeric comparisons. They allow you to test the state of your system, the characteristics of a file, date and time information, or the result of a calculation. You may want to review the variables and variable functions when determining the best way to set up an IF test.

This batch file fragment tests for a string value:

```
input "Enter your selection : " %%cmd
if "%cmd" == "WP" goto wordproc
if "%cmd" NE "GRAPHICS" goto badentry
```

This example calls *GO.BTM* if the first two characters in the file *MYFILE* are "GO" (enter this example on one line):

```
if "%@left[2,%@line[myfile,0]]" == "GO" call go.btm
```

Status Tests

These conditions test the system or command processor status. You can use internal variables and variable functions to test many other parts of the system status.

DEFINED variable

If the variable exists in the environment, the condition is true. This is equivalent to testing whether the variable is not empty, for example the following two commands are equivalent:

```
if defined abc echo Hello
if "%abc" != "" echo Hello
```

ERRORLEVEL [operator] n

This test retrieves the exit code of the preceding external program. By convention, programs return an exit code of 0 when they are successful and a number between 1 and 255 to indicate an error (depending on the program you are running, the maximum return value may be larger). The condition can be any of the operators listed above (**EQ**, **!=**, **GT**, etc.). If no operator is specified, the default is **GE**. The comparison is done numerically.

Not all programs return an explicit exit code. For programs which do not, the behavior of ERRORLEVEL is undefined.

EXIST filename

If the file exists, the condition is true. You can use wildcards in the filename, in which case the condition is true if any file matching the wildcard name exists.

Do not use IF EXIST to test for existence of a directory (use IF ISDIR instead). Due to variations in operating system

	internals, IF EXIST will not return consistent results when used to test for the existence of a directory.
ISALIAS aliasname	If the name is defined as an alias, the condition is true.
ISDIR path	If the subdirectory exists, the condition is true.
ISINTERNAL command	If the specified command is an active internal command, the condition is true. Commands can be activated and deactivated with the SETDOS /I command.
ISLABEL label	If the specified batch file label exists, the condition is true. Labels may be one or more words long.
ISWINDOW "title"	If the window which matches the title exists, the condition is true. Double quotes must be used around the title, which may contain wildcards and extended wildcards.

The first batch file fragment below tests for the existence of *A:\JAN.DOC* before copying it to drive C (this avoids an error message if the file does not exist):

```
if exist a:\jan.doc copy a:\jan.doc c:\
```

This example tests the exit code of the previous program and stops all batch file processing if an error occurred:

```
if errorlevel == 0 goto success
echo "External Error -- Batch File Ends!"
cancel
```

Combining Tests

You can negate the result of any test with **NOT**, and combine tests of any type with **.AND.**, **.OR.**, and **.XOR.**.

When two tests are combined with **.AND.**, the result is true if both individual tests are true. When two tests are combined with **.OR.**, the result is true if either (or both) individual tests are true. When two tests are combined with **.XOR.**, the result is true only if one of the tests is true and the other is false.

Test conditions are always scanned from left to right – there is no implied order of precedence, as there is in some programming languages. You can, however, force a specific order of testing by grouping conditions with parentheses, which can be nested. For example (enter this on one line):

```
if (%a == 1 .or. (%b == 2 .and. %c == 3)) echo something
```

Parentheses can only be used when the portion of the *condition* inside the parentheses contains at least one ".and.", ".or.", or ".xor.". Parentheses on a simple condition which does not combine two or more tests will be taken as part of the string to be tested, and will probably make the test fail. For example, the first of these IF tests would fail; the second would succeed:

```
if (a == a) ...
if (a == a .and. b == b) ...
```

Options

/I (Ignore case) This option is included only for compatibility with *CMD.EXE*. It has no effect, since all string comparisons under 4NT are case-insensitive.

IFF

Purpose: Perform IF / THEN / ELSE conditional execution of commands.

Format: **IFF [NOT] *condition* [.AND. | .OR. | .XOR. [NOT] *condition* ...] THEN & *commands***

[ELSEIFF *condition* THEN & *commands*] ...

[ELSE & *commands*]

& ENDIFF

condition: A test to determine if the command(s) should be executed.
commands: One or more commands to execute if the condition(s) is true. If you use multiple commands, they must be separated by command separators or be placed on separate lines of a batch file.

See also: IF and @IF.

Usage

IFF is similar to the IF command, except that it can perform one set of *commands* when a condition or set of *conditions* is true and different *commands* when the *conditions* are false.

IFF can also execute multiple commands when the *conditions* are true or false; IF normally executes only one command. IFF imposes no limit on the number of commands and is generally a "cleaner" and more structured command than IF.

IFF is always followed by one or more *conditions*. If they are true, the *commands* that follow the word THEN are executed. Additional *conditions* can be tested with ELSEIFF. If none of these *conditions* are true, the *commands* that follow the word ELSE are executed. After the selected *commands* (if any) are executed, processing continues after the word ENDIFF.

If you add a NOT before the condition, the THEN *commands* are executed only when the *condition* is false and the ELSE *commands* are executed only when the *condition* is true.

The *commands* may be separated by command separators, or may be on separate lines of a batch file. You should include a command separator or a line break after a THEN, before an ELSEIFF, and before and after an ELSE.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can group conditions with parentheses. You can nest IFF statements up to 15 levels deep. The *conditions* can test strings or numbers, the existence of a file or subdirectory, the errorlevel returned from the preceding external command, and the existence of alias names and internal commands.

See the IF command for a list of the possible *conditions*, and details on using **.AND.**, **.OR.**, **.XOR.**, and parentheses.

The *commands* can include any internal command, alias, external command, or batch file.

The alias in this example checks to see if the argument is a subdirectory. If so, the alias deletes the subdirectory's files and removes it (enter this on one line):

```
[c:\] alias prune `iff isdir %1 then & del /sxz %1 & else & echo Not a
   directory! & endiff`
```

Be sure to read the cautionary notes about GOTO and IFF under the GOTO command before using a GOTO inside an IFF statement.

If you pipe data to an IFF, the data will be passed to the command(s) following the IFF, not to IFF itself.

INKEY

Purpose: Get a single keystroke from the user and store it in an environment variable.

Format: **INKEY [/C /D /K"keys" /P /Wn /X] [*prompt*] *%%varname***

prompt: Optional text that is displayed as a prompt.
varname: The variable that will hold the user's keystroke.

/C(lear buffer)	**/P**(assword)
/D(igits only)	**/W**(ait)
/K (valid keystrokes)	**/X** (no carriage return)

See also: INPUT.

Usage

INKEY optionally displays a prompt. Then it waits for a specified time or indefinitely for a keystroke, and places the keystroke into an environment variable. It is normally used in batch files and aliases to get a menu choice or other single-key input. Along with the INPUT command, INKEY allows great flexibility in reading input from within a batch file or alias.

If *prompt* text is included in an INKEY command, it is displayed while INKEY waits for input.

The following batch file fragment prompts for a character and stores it in the variable *NUM*:

```
inkey Enter a number from 1 to 9:  %%num
```

INKEY reads standard input for the keystroke, so it will accept keystrokes from a redirected file. You can supply a list of valid keystrokes with the **/K** option.

Standard keystrokes with ASCII values between 1 and 255 are stored directly in the environment variable. Extended keystrokes (for example, function keys and cursor keys) are stored as a string in decimal format, with a leading @ (for example, the **F1** key is @59). The **Enter** key is stored as an extended keystroke, with the code @28. See ASCII and Key Codes for a list of the ASCII and extended key codes.

To test for a non-printing ASCII keystroke returned by INKEY use the @ASCII function to get the numeric value of the key. For example, to test for **Esc**, which has an ASCII value of 27:

```
inkey Enter a key:  %%key
if "%@ascii[%key]" == "27" echo Esc pressed
```

If you press **Ctrl-C** or **Ctrl-Break** while INKEY is waiting for a key, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file can handle **Ctrl-C** and **Ctrl-Break** itself with the ON BREAK command.

INKEY works within the command line window. If you prefer to use a dialog for user input, see the MSGBOX command.

Options

/C (Clear buffer) Clears the keyboard buffer before INKEY accepts keystrokes. If you use this option, INKEY will ignore any keystrokes which you type, either accidentally or intentionally, before it is ready to accept input.

/D (Digits only) Prevents INKEY from accepting any keystroke except a digit from 0 to 9.

/K"keys" Specify the permissible keystrokes. The list of valid keystrokes should be enclosed in double quotes. For alphabetic keys the validity test is not case sensitive. You can specify extended keys by enclosing their names in square brackets (within the quotes), for example:

```
inkey /k"ab[Alt-F10]" Enter A, B, Alt-F10 %%var
```

See Keys and Key Names for a complete listing of the key names you can use within the square brackets, and a description of the key name format.

If an invalid keystroke is entered, 4NT will echo the keystroke if possible, beep, move the cursor back one character, and wait for another keystroke.

/P (Password) Prevents INKEY from echoing the character.

/W (Wait) Time-out period, in seconds, to wait for a response. If no keystroke is entered by the end of the time-out period, INKEY returns with the variable unchanged. This allows you to continue the batch file if the user does not respond in a given period of time. You can specify **/W0** to return immediately if there are no keys waiting in the keyboard buffer.

For example, the following batch file fragment waits up to 10 seconds for a character, then tests to see if a "Y" was entered:

```
set net=N
inkey /K"YN" /w10 Load network (Y/N)?  %%net
iff "%net" == "Y" then
      rem Commands to load the network go here
endiff
```

/X (No carriage return) Prevents INKEY from displaying a carriage return and line feed after the user's entry.

INPUT

Purpose: Get a string from the keyboard and save it in an environment variable.

Format: **INPUT [/C /D /E /Ln /N /P /Wn /X] [*prompt*] *%%varname***

prompt: Optional text that is displayed as a prompt.
varname: The variable that will hold the user's input.

/C(lear buffer)	**/N**(o colors)
/D(igits only)	**/P**(assword)
/E(dit)	**/W**(ait)
/L(ength)	**/X** (no carriage return)

See also: INKEY.

Usage

INPUT optionally displays a prompt. Then it waits for a specified time or indefinitely for your entry. It places any characters you type into an environment variable. INPUT is normally used in batch files and aliases to get multi-key input. Along with the INKEY command, INPUT allows great flexibility in reading user input from within a batch file or alias.

If *prompt* text is included in an INPUT command, it is displayed while INPUT waits for input. Standard command-line editing keys may be used to edit the input string as it is entered. If you use the **/P** password option, INPUT will echo asterisks instead of the keys you type.

All characters entered up to, but not including, the carriage return are stored in the variable.

The following batch file fragment prompts for a string and stores it in the variable FNAME:

```
input Enter the file name:  %%fname
```

INPUT reads standard input, so it will accept text from a re- directed file.

If you press **Ctrl-C** or **Ctrl-Break** while INPUT is waiting for input, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file can handle **Ctrl-C** and **Ctrl-Break** itself with the ON BREAK command.

You can pipe text to INPUT; if you do, it will set the variable to the first line it receives. However, this variable will be set in the "child" process used to handle the right hand side of the pipe, and it will not be available in the original copy of 4NT used to start the pipe.

INPUT works within the command line window. If you prefer to use a dialog for user input, see the MSGBOX command.

Options

/C (Clear buffer) Clears the keyboard buffer before INPUT accepts keystrokes. If you use this option, INPUT will ignore any keystrokes which you type, either accidentally or intentionally, before INPUT is ready.

/D (Digits only) Prevents INKEY from accepting any keystroke except a digit from 0 to 9.

/E (Edit) Allows you to edit an existing value. If there is no existing value for *varname*, INPUT proceeds as if **/E** had not been used, and allows you to enter a new value.

/Ln (Length) Sets the maximum number of characters which INPUT will accept to "n". If you attempt to enter more than this number of characters, INPUT will beep and prevent further input (you will still be able to edit the characters typed before the limit was reached).

/N (No colors) Disables the use of input colors defined in the InputColor directive in the *4NT.INI* file, and forces INPUT to use the default display colors.

/P (Password) Tells INPUT to echo asterisks, instead of the characters you type.

/W (Wait) Time-out period, in seconds, to wait for a response. If no keystroke is entered by the end of the time-out period, INPUT returns with the variable unchanged. This allows you to continue the batch file if the user does not respond in a given period of time. If you enter a key before the time-out period, INPUT will wait indefinitely for the remainder of the line. You can specify **/W0** to return immediately if there are no keys waiting in the keyboard buffer.

/X (No carriage return) Prevents INKEY from displaying a carriage return and line feed after the user's entry.

KEYBD

Purpose: Set the state of the keyboard toggles: Caps Lock, Num Lock, and Scroll Lock.

Format: **KEYBD [/Cn /Nn /Sn]**

n: 0 to turn off the toggle, or 1 to turn on the toggle.

/C(aps lock) **/S**(croll lock)
/N(um lock)

Usage

Most keyboards have 3 toggle keys, the Caps Lock, Num Lock, and Scroll Lock. The toggle key status is usually displayed by three lights at the top right corner of the keyboard.

This command lets you turn any toggle key on or off. It is most useful in batch files and aliases if you want the keys set a particular way before collecting input from the user.

For example, to turn off the Num Lock and Caps Lock keys, you can use this command:

```
[c:\] keybd /c0 /n0
```

If you use the KEYBD command with no switches, it will display the present state of the toggle keys.

In Windows NT, the toggle key state is the same for each session. Changes made with KEYBD will therefore affect all other sessions.

Options

/C (Caps lock) Turn the Caps Lock key on or off.

/N (Num lock) Turn the Num Lock key on or off.

/S (Scroll lock) Turn the Scroll Lock key on or off.

KEYS

Purpose: Enable, disable, or display the history list.

Format: **KEYS [ON | OFF | LIST]**

See also: HISTORY.

Usage

This command is provided for compatibility with the KEYS command in *CMD.EXE*, which controls the history list in Windows NT. The same functions are available by setting the HistMin directive in the *.INI* file, and by using the HISTORY command.

The history list collects the commands you type for later recall, editing, and viewing. You can view the contents of the list through the history list window or by typing any of the following commands:

```
[c:\] history
[c:\] history /p
[c:\] keys list
```

The first command displays the entire history list. The second displays the entire list and pauses at the end of each full screen. The third command produces the same output as the first, except that each line is numbered.

You can disable the collection and storage of commands in the history list by typing:

```
[c:\] keys off
```

You can turn the history back on with the command:

```
[c:\] keys on
```

If you issue the KEYS command without any parameters, 4NT will show you the current status of the history list.

LIST

Purpose: Display a file, with forward and backward paging and scrolling.

Format: **LIST [/A:[[-]rhsda] /H /I /R /S /T /W /X] *file*...**

file: A file or list of files to display.

/A: (Attribute select)	**/S**(tandard input)
/H(igh bit off)	**/T** (search for Text)
/I(gnore wildcards)	**/W**(rap)
/R(everse)	**/X** (heX display mode)

See also: TYPE.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

LIST provides a fast and flexible way to view a file, without the overhead of loading and using a text editor.

For example, to display a file called *MEMO.DOC*:

```
[c:\] list memo.doc
```

LIST is most often used for displaying ASCII text files. It can be used for other files which contain non-alphabetic characters, but you may need to use hex mode (see below) to read these files.

LIST uses the cursor pad to scroll through the file. The following keys have special meanings:

Home	Display the first page of the file.
End	Display the last page of the file.
Esc	Exit the current file.
Ctrl-PgUp	Display previous file.
Ctrl-PgDn	Display next file.
Ctrl-C	Quit LIST.
	Scroll up one line.
↓	Scroll down one line.
PgUp	Scroll up one page.
PgDn or	
Space	Scroll up one page.
←	Scroll left 8 columns.
→	Scroll right 8 columns.
Ctrl ←	Scroll left 40 columns.

Ctrl →	Scroll right 40 columns.
F1	Display online help
B	Go back one file to the previous file in the current group of files.
F	Prompt and search for a string.
Ctrl-F	Prompt and search for a string, searching backward from the end of the file.
G	Go to a specific line or, in hex mode, to a specific hexadecimal offset.
H	Toggle the "strip high bit" (**/H**) option.
I	Display information on the current file (the full name, size, date, and time).
N	Find next matching string.
Ctrl-N	Find previous match in the file.
P	Print the current page or the entire file.
W	Toggle the "line wrap" (**/W**) option.
X	Toggle the hex-mode display (**/X**) option.

Text searches performed with **F**, **N**, **Ctrl-F**, and **Ctrl-N** are not case sensitive. However, if the display is currently in hexadecimal mode and you press **F** or **Ctrl-F**, you will be prompted for whether you want to search in hexadecimal mode. If you answer **Y**, you should then enter the search string as a sequence of 2-digit hexadecimal numbers separated by spaces, for example **41 63 65** (these are the ASCII values for the string "Ace"). Hexadecimal searches **are** case sensitive, and search for exactly the string you enter.

When the search string is found LIST displays the line containing the string at the top of the screen, and highlights the string it found. Any additional occurrences of the string on the same display page are also highlighted. Highlighting is intended for use with text files; in binary files the search string will be found, but may not be highlighted properly.

You can use wildcards in the search string. For example, you can search for the string "to*day" to find the next line which contains the word "to" followed by the word "day" later on the same line, or search for the numbers "101" or "401" with the search string "[14]01". If you begin the search string with a back-quote [`], or enclose it in back-quotes, wildcard characters in the string will be treated as normal text with no special wildcard meaning.

LIST saves the search string used by **F**, **N**, **Ctrl-F**, and **Ctrl-N**, so you can LIST multiple files and search for the same string simply by pressing **N** in each file, or repeat your search the next time you use LIST.

You can use the **/T** switch to specify search text for the first *file*. When you do so, LIST begins a search as soon as the file is loaded. Use **/I** to ignore wildcards in the initial search string, and **/R** to make the initial search go backwards from the end of the file. When you LIST multiple files with a single LIST command, these switches affect only the first file; they are ignored for the second and subsequent files.

LIST normally allows long lines in the file to extend past the right edge of the screen. You can use the horizontal scrolling keys (see above) to view text that extends beyond the screen width. If you use the **W** command or **/W** switch to wrap the display, each line is wrapped when it reaches the right edge of the screen, and the horizontal scrolling keys are disabled.

To view text from the clipboard, use **CLIP:** as the file to be listed. CLIP: will not return any data unless the clipboard contains text. See Redirection for additional information on CLIP:.

You can use **G** to go to a specific line or hexadecimal offset. When prompted for a line number you can enter a negative number to go backward a specified number of lines from the current position (there is no corresponding option to go forward a certain number of lines). When you use this option the number of lines moved will only correspond to the line count in the status bar if the display is **not** wrapped.

If you print the file which LIST is displaying, you will be asked whether you wish to print the entire file or the current display page. The print format will match the display format. If you have switched to hexadecimal or wrapped mode, that mode will be used for the printed output as well. If you print in wrapped mode, long lines will be wrapped at the width of the display. If you print in normal display mode without line wrap, long lines will be wrapped or truncated by the printer, not by LIST.

Printed output normally goes to device LPT1. If you wish to send the printed output to another device, use the Commands page of the OPTION dialogs, or the Printer directive in the *.INI* file.

If you specify a directory name instead of a filename as an argument, LIST will display each of the files in that directory.

Most of the LIST keystrokes can be reassigned with *.INI* file key mapping directives.

You can set the colors used by LIST with on the Commands page of the OPTION dialogs, or ListColors and ListStatBarColors directives in the *.INI* file. If ListColors is not used, the LIST display will use the current default colors. If ListStatBarColors is not used, the status bar will use the reverse of the LIST display colors.

By default, LIST sets tab stops every 8 columns. You can change this behavior on the Display page of the OPTION dialogs, or with the TabStops *.INI* file directive

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **LIST /A: ...**), LIST will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/H (High bit off) Strip the high bit from each character before displaying. This is useful when displaying files created by some word processors that turn on the high bit for formatting purposes. You can toggle this option on and off from within LIST with the **H** key.

/I (Ignore wildcards) Only meaningful when used in conjunction with the **/T "text"** option. Directs LIST to interpret characters such as *****, **?**, **[**, and **]** as literal characters instead of wildcard characters. **/I** affects only the initial search started by **/T**, not subsequent searches started from within LIST.

/R (Reverse) Only meaningful when used in conjuction with the **/T "text"** option. Directs LIST to search for text from the end of the file instead of from the beginning of the file. Using this switch can speed up searches for text that is normally near the end of the file, such as a signature. **/R** affects only the initial search started by **/T**, not subsequent searches started from within LIST.

/S (Standard input) Read from standard input rather than a file. This allows you to redirect command output and view it with LIST. Normally, LIST will detect input from a redirected command and adjust automatically. However, you may find circumstances when **/S** is required. For example, to use LIST to display the output of DIR you could use either of these commands:

```
[c:\] dir | list
[c:\] dir | list /s
```

/T (Text) Search for text in the first *file*. This option is the same as pressing **F**, but it allows you to specify the search text on the command line. The text must be contained in quotation marks if it contains spaces, punctuation, or wildcard characters. For example, to search for the string 4NT in the file *README.DOC*, you can use this command:

```
[c:\] list /t4nt readme.doc
```

The search text may include wildcards and extended wildcards. For example, to search for the words Hello and John on the same line in the file *LETTER.DAT*:

```
[c:\] list /t"Hello*John" letter.dat
```

When you LIST multiple files with a single LIST command, **/T** only initiates a search in the first file. It is ignored for the second and subsequent files. Also see **/I** and **/R**.

/W (Wrap) Wrap the text at the right edge of the screen. This option is useful when displaying files that don't have a carriage return at the end of each line. The horizontal scrolling keys do not work when the display is wrapped. You can toggle this option on and off from within LIST with the **W** key.

/X (hex mode) Display the file in hexadecimal (hex) mode. This option is useful when displaying executable files and other files that contain non-text characters. Each byte of the file is shown as a pair of hex characters. The corresponding text is displayed to the right of each line of hexadecimal data. You can toggle this mode on and off from within LIST with the **X** key.

LOADBTM

Purpose: Switch a batch file to or from BTM mode.

Format: **LOADBTM [ON | OFF]**

Usage

4NT recognizes two kinds of batch files: *.BAT* or *.CMD*, and *.BTM*. Batch files executing in BTM mode run two to ten times faster than in CMD or BAT mode. Batch files automatically start in the mode indicated by their extension.

The LOADBTM command turns BTM mode on and off. It can be used to switch modes in either a *.BAT*, *.CMD* or *.BTM* file. If you use LOADBTM with no argument, it will display the current batch mode: LOADBTM ON or LOADBTM OFF.

Using LOADBTM to repeatedly switch modes within a batch file is not efficient. In most cases the speed gained by running some parts of the file in BTM mode will be more than offset by the speed lost through repeated loading of the file each time BTM mode is invoked.

LOADBTM can only be used within a batch file. It is most often used to convert a *.CMD* or *.BAT* file to BTM mode without changing its extension.

Using LOADBTM to repeatedly switch modes within a batch file is not efficient. In most cases the speed gained by running some parts of the file in BTM mode will be more than offset by the speed lost through repeated loading of the file each time BTM mode is invoked.

LOG

Purpose: Save a log of commands to a disk file.

Format: **LOG [/H /W *file*] [ON | OFF | *text*]**

file: The name of the file to hold the log.
text: An optional message that will be added to the log.

/H(istory log) **/W**(rite to)

See also: HISTORY.

Usage

LOG keeps a record of all internal and external commands you use, whether they are executed from the prompt or from a batch file. Each entry includes the current system date and time, along with the actual command after any alias or variable expansion. You can use the log file as a record of your daily activities.

LOG with the **/H** option keeps a similar record called a "history log". The history log records only commands entered at the prompt; it does not record batch file commands. In addition, the history log does not record the date and time for each command, and it records commands before aliases and variables are expanded.

The two logging options are independent. You can have both a regular log and a history log enabled simultaneously.

By default, LOG writes to the file *4NTLOG* in the root directory of the boot drive. The default file name for the history log is *4NTHLOG*. You can set the default log file names on the Options 2 page of the OPTION dialogs, or with the LogName and HistLogName directives in the *.INI* file.

Entering LOG or LOG /H with no parameters displays the name of the log file and the log status (ON or OFF):

```
[c:\] log
LOG (C:\4NTLOG) is OFF
```

To enable or disable logging, add the word "ON" or "OFF" after the LOG command:

```
[c:\] log on
```

or

```
[c:\] log /h on
```

Entering LOG or LOG /H with *text* writes a message to the log file, even if logging is set OFF. This allows you to enter headers in the log file:

```
[c:\] log "Started work on the database system"
```

The LOG file format looks like this:

```
[date  time]  command
```

where the date and time are formatted according to the country code set for your system.

The LOG /H output can be used as the basis for writing batch files. Start LOG /H, then execute the commands that you want the batch file to execute. When you are finished, turn LOG /H off. The resulting file can be turned into a batch file that performs the same commands with little or no editing.

You can have both a regular log (with time and date stamping) and a history log (without the time stamps) enabled simultaneously.

Options

/H (History log) This option makes the other options on the command line (after the **/H**) apply to the history log. For example, to turn on history logging and write to the file C:\LOG\HLOG:

```
[c:\]  log /h /w c:\log\hlog
```

/W (Write) This switch specifies a different filename for the LOG or LOG /H output. It also automatically performs a LOG ON command. For example, to turn logging on and write the log to *C:\LOG\LOGFILE*:

```
[c:\] log /w c:\log\logfile
```

Once you select a new file name with the LOG /W or LOG /H/W command, LOG will use that file until you issue another LOG /W or LOG /H/W command, or until you reboot your computer. Turning LOG or LOG /H off or on does not change the file name.

MD / MKDIR

Purpose: Create a subdirectory.

Format: **MD [/N /S] *path*...**

or

MKDIR [/N /S] *path*...

path: The name of one or more directories to create.

/N(o update) **/S**(ubdirectories)

See also: RD.

Usage

MD and MKDIR are synonyms. You can use either one.

MD creates a subdirectory anywhere in the directory tree. To create a subdirectory from the root, start the *path* with a backslash [****]. For example, this command creates a subdirectory called *MYDIR* in the root directory:

```
[c:\] md \mydir
```

If no path is given, the new subdirectory is created in the current directory. This example creates a subdirectory called *DIRTWO* in the current directory:

```
[c:\mydir] md dirtwo
```

To create a directory from the parent of the current directory (that is, to create a sibling of the current directory), start the pathname with two periods and a backslash [**..**].

When creating a directory on an HPFS, NTFS, or LFN drive, you must quote any *path* which contains whitespace or special characters. See File Names for additional details.

If MD creates one or more directories, they will be added automatically to the extended directory search database unless the **/N** option is specified.

Options

/N (No update) Do not update the extended directory search database, *JPSTREE.IDX*. This is useful when creating a temporary directory which you do not want to appear in the extended search database.

/S (Subdirectories) MD creates one directory at a time unless you use the **/S** option. If you need to create the directory *C:\ONE\TWO\THREE* and none of the named directories exist, you can use **/S** to have MD create all of the necessary subdirectories for you in a single command:

```
[c:\] md /s \one\two\three
```

For compatibility with Windows NT's *CMD.EXE*, **/S** becomes the default if you enable command processor extensions with the **/X** switch on the 4NT startup command line.

MEMORY

Purpose: Display the amount and status of system memory.

Format: **MEMORY**

Usage

MEMORY lists the percentage "memory load" as reported by Windows NT, the total and available physical RAM, the total and available page file size, the total and free environment and alias space, and the total history space.

The memory load is a figure returned by the operating system which gives an overall sense of memory utilization. It is not a precise indicator of system load or memory usage. The total page file figure shows the total number of bytes that can be stored in the file, but may not reflect the actual size of the current file on disk.

MOVE

Purpose: Move files to a new directory and drive.

Format: **MOVE [/A:[[-]rhsda] /C /D /E /H /M /N /P /Q /R /S /T /U /V]** ***source...*** ***destination***

source: A file or list of files to move.
destination: The new location for the files.

/A: (Attribute select)	**/P**(rompt)
/C(hanged)	**/Q**(uiet)
/D(irectory)	**/R**(eplace)
/E (no error messages)	**/S**(ubdirectory tree)
/H(idden and system)	**/T**(otal)
/M(odified files)	**/U**(pdate)
/N(othing)	**/V**(erify)

See also: COPY and RENAME.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, or size ranges anywhere on the line apply to all *source* files.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

The MOVE command moves one or more files from one directory to another, whether the directories are on the same drive or not. It has the same effect as copying the files to a new location and then deleting the originals. Like COPY and RENAME, MOVE works with single files, multiple files, and sets of files specified with an include list.

The simplest MOVE command moves a single *source* file to a new location and, optionally, gives it a new name. These two examples both move one file from drive *C:* to the root directory on drive *A:*

```
[c:\] move myfile.dat a:\
[c:\] move myfile.dat a:\savefile.dat
```

In both cases, *MYFILE.DAT* is removed from drive *C:* after it has been copied to drive *A:*. If a file called *MYFILE.DAT* in the first example, or *SAVEFILE.DAT* in the second example, already existed on drive *A:*, it would be overwritten. (This demonstrates the difference between MOVE and RENAME. MOVE will move files between drives and will overwrite the destination file if it exists; RENAME will not.)

When you move a single file, the *destination* can be a directory name or a file name. If it is a directory name, and you add a backslash [\] to the end of the name, MOVE will display an error message if the name does not refer to an existing directory. You can use this feature to keep MOVE from treating a mistyped *destination* directory name as a file name, and attempting to move the *source* file to that name.

If you move multiple files, the *destination* **must** be a directory name. MOVE will move each file into the *destination* directory with its original name. If the *destination* is not a directory, MOVE will display an error message and exit. For example, if *C:\FINANCE\MYFILES* is not a directory, this command will display an error; otherwise, the files will be moved to that directory:

```
[c:\] move *.wks *.txt c:\finance\myfiles
```

The **/D** option can be used for single or multiple file moves; it checks to see whether the *destination* is a directory, and will prompt to see if you want to create the *destination* directory if it doesn't exist.

If MOVE creates one or more destination directories, they will be added automatically to the extended directory search database.

You cannot move a file to a character device like the printer, or to itself.

Be careful when you use MOVE with the SELECT command. If you SELECT multiple files and the destination is not a directory (for example, because of a misspelling), MOVE will assume it is a file name. In this case each file will be moved in turn to the destination file, overwriting the previous file, and then the original will be erased before the next file is moved. At the end of the command, all of the original files will have been erased and only the last file will exist as the destination file.

You can avoid this problem by using square brackets with SELECT instead of parentheses (be sure that you don't allow the command line to get too long -- watch the character count in the upper left corner while you're selecting files). MOVE will then receive one list of files to move instead of a series of individual filenames, and it will detect the error and halt. You can also add a backslash [\] to the end of the *destination* name to ensure that it is the name of a subdirectory (see above).

Advanced Features and Options

MOVE first attempts to rename the file(s), which is the fastest way to move files between subdirectories on the same drive. If that fails (*e.g.*, because the *destination* is on a different drive or already exists), MOVE will copy the file(s) and then delete the originals.

If MOVE must physically copy the files and delete the originals, rather than renaming them (see above), then some disk space may be freed on the *source* drive. The free space may be the result of moving the files to another drive, or of overwriting a larger *destination* file with a smaller *source* file. MOVE displays the amount of disk space recovered unless the **/Q** option is used (see below). It does so by comparing the amount of free disk space before and after the MOVE command is executed. However, this amount may be incorrect if you are using a deletion tracking system which retains deleted files for later recovery, or if another program performs a file operation while the MOVE command is executed.

When physically copying files, MOVE preserves the hidden, system, and read-only attributes of the *source* files, and sets the archive attribute of the *destination* files. However, if the files can be renamed, and no copying is required, then the file attributes are not changed. See File Attributes and Time Stamps.

Use caution with the **/A:** and **/H** switches (both of which can allow MOVE to process hidden files) when you are physically moving files, and both the *source* and *destination* directories contain file descriptions. If the *source* file specification matches the description file name (normally *DESCRIPT.ION*), and you tell MOVE to process hidden files, the *DESCRIPT.ION* file itself will be moved, overwriting any existing file descriptions in the *destination* directory. For example, if the *C:\DATA* directory contains file descriptions this command would overwrite any existing descriptions in the *D:\SAVE* directory:

```
[c:\data] move /h d*.* d:\save\
```

(If you remove the hidden attribute from the *DESCRIPT.ION* file the same caution applies even if you do not use **/A:** or **/H**, as *DESCRIPT.ION* is then treated like any other file.)

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **MOVE /A: ...**), MOVE will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

See the cautionary note under **Advanced Features and Options** above before using **/A:** when both *source* and *destination* directories contain file descriptions.

/C (Changed files) Move files only if the *destination* file exists and is older than the *source* (see also **/U**). This option is useful for updating the files in one directory from those in another without moving any newly-created files.

/D (Directory) Requires that the *destination* be a directory. If the *destination* does not exist, MOVE will prompt to see if you want to create it. If the *destination* exists as a file, MOVE will fail with an "Access denied" error. Use this option to avoid having MOVE accidentally interpret your *destination* name as a file name when it's really a mistyped directory name.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/H (Hidden) Move all files, including hidden and system files. See the cautionary note under **Advanced Features and Options** above before using **/H** when both *source* and *destination* directories contain file descriptions.

/M (Modified files) Move only files that have the archive bit set. The archive bit will remain set after the MOVE; to clear it use ATTRIB.

/N (Nothing) Do everything except actually move the file(s). This option is most useful for testing what a complex MOVE command will do.

/P (Prompt) Prompt the user to confirm each move. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames, the total number of files moved, or the amount of disk space recovered, if any. This option is most often used in batch files. See also **/T**.

/R (Replace) Prompt for a **Y** or **N** response before overwriting an existing *destination* file.

/S (Subdirectories) Move an entire subdirectory tree to another location. MOVE will attempt to create the *destination* directories if they don't exist, and will remove empty subdirectories after the move. When **/D** is used with **/S**, you will be prompted if the first *destination* directory does not exist, but subdirectories below that will be created automatically by MOVE. If MOVE /S creates one or more destination directories, they will be added automatically to the extended directory search database.

If you attempt to use **/S** to move a subdirectory tree into part of itself, MOVE will detect the resulting infinite loop, display an error message, and exit.

/T (Total) Don't display filenames as they are moved, but display the total number of files moved and the amount of free disk space recovered, if any.

/U (Update) Move each *source* file only if it is newer than a matching *destination* file or if a matching *destination* file does not exist (also see **/C**). This option is useful for moving new or changed files from one directory to another.

/V (Verify) Verify each disk write. This is the same as executing the VERIFY ON command, but is only active during the MOVE. **/V** does **not** read back the file and compare its contents with what was written; it only verifies that the data written to disk is physically readable.

MSGBOX

Purpose: Display a message box and return the user's response.

Format: **MSGBOX OK | OKCANCEL | YESNO | YESNOCANCEL ["title"] *prompt***

title: Text for the title bar of the message box.
prompt: Text that will appear inside the message box.

See also: INKEY and INPUT.

Usage

MSGBOX can display one of 4 kinds of message boxes and wait for the user's response. You can use ***title*** and ***prompt*** to display any text you wish. 4NT automatically sizes and locates the box on the screen.

The message box may have 1, 2, or 3 response buttons. The command MSGBOX OK creates a single-button box; the user must simply acknowledge the prompt text.

The OKCANCEL and YESNO forms have 2 buttons each. The YESNOCANCEL form has 3 buttons. The button the user chooses is returned in the 4NT variable **%_?**. Be sure to save the return value in another variable or test it immediately, because the value of **%_?** changes with every internal command.

The following list shows the value returned for each possible selection:

Yes	10	**No**	11
OK	10	**Cancel**	12

If you exit the message box without selecting one of these options (for example, some message boxes allow you to exit by pressing **Esc** or double-clicking the close button), MSGBOX will set **%_?** to 0. If there is an error in the MSGBOX command itself, **%_?** will be set to 1 for a syntax error or 2 for any other error.

For example, to display a Yes or No message box and take action depending on the result, you could use commands like this:

```
msgbox yesno "Copy" Copy all files to A:?
iff %_? == 10 then
     copy *.* a:
endiff
```

MSGBOX creates a popup dialog box. If you prefer to retrieve input from inside the command line window, see the INKEY and INPUT commands.

ON

Purpose: Execute a command in a batch file when a specific condition occurs.

Format: **ON BREAK [*command*]**

or

ON ERROR [*command*]

or

ON ERRORMSG [*command*]

Usage

ON can only by used in batch files.

ON sets a "watchdog" that remains in effect for the duration of the current batch file. Whenever a BREAK or ERROR condition occurs after ON has been executed, the corresponding *command* is automatically executed.

ON BREAK will execute the *command* if the user presses **Ctrl- C** or **Ctrl-Break**.

ON ERROR and ON ERRORMSG will execute the *command* after any command processor or operating system error (including critical errors). That is, they will detect errors such as a disk write error, and 4NT errors such as a COPY command that fails to copy any files, or the use of an unacceptable command option.

ON ERROR executes the *command* immediately after the error occurs, without displaying any command processor error message (operating system errors may still be displayed in some cases). ON ERRORMSG displays the appropriate error message, then executes the *command*. If both are specified, ON ERROR will take precedence, and the ON ERRORMSG setting will be ignored. The remainder of this section discusses both settings together, using the term "ON ERROR[MSG]".

ON BREAK and ON ERROR[MSG] are independent of each other. You can use either one, or both, in any batch file.

Each time ON BREAK or ON ERROR[MSG] is used, it defines a new *command* to be executed for a break or error, and any old *command* is discarded. If you use ON BREAK or ON ERROR[MSG] with no following *command*, that type of error handling is disabled. Error handling is also automatically disabled when the batch file exits.

ON BREAK and ON ERROR[MSG] only affect the current batch file. If you CALL another batch file, the first batch file's error handling is suspended, and the CALLed file must define its own error handling. When control returns to the first batch file, its error handling is reactivated.

The *command* can be any command that can be used on a batch file line by itself. Frequently, it is a GOTO or GOSUB command. For example, the following fragment traps any user attempt to end the batch file by pressing **Ctrl-C** or **Ctrl-Break**. It scolds the user for trying to end the batch file and then continues displaying the numbers from 1 to 1000:

```
on break gosub gotabreak
do i = 1 to 1000
   echo %i
enddo
quit
```

```
        :gotabreak
        echo Hey!  Stop that!!
        return
```

You can use a command group as the *command* if you want to execute multiple commands, for example:

```
        on break (echo Oops, got a break! & quit)
```

ON BREAK and ON ERROR[MSG] always assume that you want to continue executing the batch file. After the *command* is executed, control automatically returns to the next command in the batch file (the command after the one that was interrupted by the break or error).   The only way to avoid continuing the batch file after a break or error is for the *command* to transfer control to another point with GOTO, end the batch file with QUIT or CANCEL, or start another batch file (without CALLing it).

When handling an error condition with ON ERROR[MSG], you may find it useful to use internal variables, particularly %_? and %_SYSERR, to help determine the cause of the error.

The ON ERROR[MSG] command will **not** be invoked if an error occurs while reading or writing a redirected input file, output file, or pipe.

Caution: If a break or error occurs while the *command* specified in ON BREAK or ON ERROR[MSG] is executing, the *command* will be restarted. This means you must use caution to avoid or handle any possible errors in the commands invoked by ON ERROR[MSG], since such errors can cause an infinite loop.

OPTION

Purpose: Modify the command processor configuration.

Format: **OPTION [//optname=value ...]**

optname: An INI file directive to set or modify.
value: A new value for that directive.

See also: *.INI* File Directives.

Usage

OPTION displays a set of dialogs which allows you to modify many of the configuration options stored in the *.INI* file.

When you exit from the dialogs, you can select **Save** to save your changes in the *.INI* file for use in the current session and all future sessions, select **Use** to use your changes in the current session only, or select **Cancel** to discard the changes.

OPTION does not preserve comments when saving modified settings in the *.INI* file. To be sure *.INI* file comments are preserved, put them on separate lines in the file (see *.INI* File Directives for details).

Save saves all changes since the last **Save**, or since the last time you started the command processor. If you run OPTION and exit with **Use** or **OK**, any changes will not be saved in the *.INI* file at that time. However, if you run OPTION again and exit with **Save**, any earlier changes will automatically be saved in the *.INI* file along with any new changes.

In most cases, changes you make in the **Startup** section of the OPTION dialogs or notebook will only take effect when you restart 4NT.

Other changes take effect as soon as you exit the dialogs or notebook with **Save** or **Use**. However, not all option changes will appear immediately, even if they have taken effect. For example, some color changes will only appear after a CLS command.

OPTION handles most standard *.INI* file settings. More advanced settings, including all those listed under Key Mapping Directives and Advanced Directives cannot be modified with the OPTION dialogs or notebook. These settings must be inserted or modified in the *.INI* file manually. For more details see Modifying the *.INI* File.

Setting Individual Options

If you follow the OPTION command with one or more sequences of a double slash mark [//] followed by an **option=value** setting, the OPTION dialogs or notebook will not appear. Instead, the new settings will take effect immediately, and will be in effect for the current session only. This example turns off batch file echo and changes the input colors to bright cyan on black (enter this all on one line):

```
[c:\] option //BatchEcho = No //InputColors = bri cya on bla
```

Option values may contain whitespace. However, you cannot enter an option value which contains the "//" string.

This feature is most useful for testing settings quickly, and in aliases or batch files which depend on certain options being in effect.

Changes made with ***//*** are temporary. They will not be saved in the *.INI* file, even if you subsequently load the option dialogs and select **Save**.

PATH

Purpose: Display or alter the list of directories that 4NT will search for executable files, batch files, and files with executable extensions that are not in the current directory.

Format: **PATH [*directory* [;*directory*...]]**

directory: The full name of a directory to include in the path setting.

See also: ESET and SET.

Usage

When 4NT is asked to execute an external command (a *.COM*, *.EXE*, *.BTM*, *.BAT*, or *.CMD* file or executable extension), it first looks for the file in the current directory. If it fails to find an executable file there, it then searches each of the *directories* specified in the PATH setting.

For example, after the following PATH command, 4NT will search for an executable file in four directories: the current directory, then the root directory on drive C, then the *DOS* subdirectory on C, and then the *UTIL* subdirectory on C:

```
[c:\] path c:\;c:\dos;c:\util
```

The list of *directories* to search is stored as an environment string, and can also be set or viewed with SET, and edited with ESET.

Directory names in the path must be separated by semicolons [**;**]. Each directory name is shifted to upper case to maintain compatibility with programs which can only recognize upper case directory names in the path. If you modify your path with the SET or ESET command, you may include directory names in lower case. These may cause trouble with some programs, which assume that all path entries have been shifted to upper case.

On drives which support long filenames, some directory names may include spaces or other special characters. Unlike other commands where quotes are required, such names should **not** be quoted in the PATH.

If you enter PATH with no parameters, the current path is displayed:

```
[c:\] path
PATH=C:\;C:\DOS;C:\UTIL
```

Entering PATH and a semicolon clears the search path so that only the current directory is searched for executable files (this is the default at system startup).

Some applications also use the PATH to search for their data files.

If you include an explicit file extension on a command name (for example, *WP.EXE*), the search will find files with that name and extension in the current directory and every directory in the path. It will not locate other executable files with the same base name (*e.g.*, *WP.COM*).

If you have an entry in the path which consists of a single period [**.**], the current directory will **not** be searched first, but instead will be searched when 4NT reaches the "**.**" in the path. This allows you to delay the search of the current directory for executable files and files with executable extensions. In rare cases, this feature may not be compatible with applications which use the path to find their files; if you experience a problem, you will have to remove the "." from the path while using any such application.

To create a path longer than the command-line length limit, use PATH repeatedly to append additional directories to the path:

```
      path [first list of directories]
      path %path;[second list of directories]          ...
```

You cannot use this method to extend the path beyond 2042 characters (the internal buffer limit, with room for "PATH "). It is usually more efficient to use aliases to load application programs than to create a long PATH. See ALIAS for details.

If you specify an invalid directory in the path, it will be skipped and the search will continue with the next directory in the path.

PAUSE

Purpose: Suspend batch file or alias execution.

Format: **PAUSE [*text*]**

text: The message to be displayed as a user prompt.

Usage

A PAUSE command will suspend execution of a batch file or alias, giving you the opportunity to change disks, turn on the printer, etc.

PAUSE waits for any key to be pressed and then continues execution. You can specify the *text* that PAUSE displays while it waits for a keystroke, or let it use the default message:

```
      Press any key when ready...
```

For example, the following batch file fragment prompts the user before erasing files:

```
      pause Press Ctrl-C to abort, any other key to erase all .LST files
      erase *.lst
```

If you press **Ctrl-C** or **Ctrl-Break** while PAUSE is waiting for a key, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job.   In a batch file you can handle **Ctrl-C** and **Ctrl-Break** yourself with the ON BREAK command.

POPD

Purpose: Return to the disk drive and directory at the top of the directory stack..

Format: **POPD [*]**

See also: DIRS, PUSHD, and Directory Navigation.

Usage

Each time you use the PUSHD command, it saves the current disk drive and directory on the internal directory stack. POPD restores the last drive and directory that was saved with PUSHD and removes that entry from the stack. You can use these commands together to change directories, perform some work, and return to the starting drive and directory.

Directory changes made with POPD are recorded in the directory history list and can be displayed in the directory history window

This example saves and changes the current disk drive and directory with PUSHD, and then restores it. The current directory is shown in the prompt:

```
[c:\] pushd d:\database\test
[d:\database\test] popd
[c:\]
```

You can use the DIRS command to see the complete list of saved drives and directories (the directory stack).

The POPD command followed by an asterisk [*****] clears the directory stack without changing the current drive and directory.

If the directory on the top of the stack is not on the current drive, POPD will switch to the drive and directory on the top of the stack without changing the default directory on the current drive.

PROMPT

Purpose:	Change the command-line prompt.
Format:	**PROMPT [*text*]**
	text: Text to be used as the new command-line prompt.

Usage

You can change and customize the command-line prompt at any time. The prompt can include normal text, and system information such as the current drive and directory, the time and date, and the amount of memory available. You can create an informal "Hello, Bob!" prompt or an official-looking prompt full of impressive information. The prompt *text* can contain special commands in the form **$?**, where **?** is one of the characters listed below:

b	The vertical bar character [**|**].
c	The open parenthesis [**(**].
d	Current date, in the format: *Fri 12-12-97*(the month, day, and year are formatted according to your current country settings).
D	Current date, in the format: *Fri Dec 12, 1997*.
e	The ASCII ESC character (decimal 27).
f	The close parenthesis [**)**].
g	The **>** character.
h	Backspace over the previous character.
i	Display the Windows NT prompt header line, which reminds you of how to return to the Windows NT desktop, or get help.
l	The **<** character.
m	Time in hours and minutes using 24-hour format.
M	Time in hours and minutes using the default country format and retaining "a" or "p"*, e.g.* 4:07p.
n	Current drive letter.
p	Current drive and directory (lower case).
P	Current drive and directory (upper case on drives which do not support long filenames; directory names shown in mixed case as stored on the disk on HPFS, NTFS, and LFN drives).
q	The **=** character.
r	The numeric exit code of the last external command.
s	The space character.
t	Current 24-hour time, in the format *hh:mm:ss*.
T	Current 12-hour time, in the format *hh:mm:ss[a|p]*.
v	Operating system version number, in the format *3.50*.
xd:	Current directory on drive *d:*, in lower case, including the drive letter. (Uses the actual case of the directory name as stored on the disk for HPFS, NTFS, and LFN drives.)

Xd:	Current directory on drive *d:*, in upper case, including the drive letter.
z	Current shell nesting level; the primary command processor is shell 0.
+	Display one **+** character for each directory on the PUSHD directory stack.
$	The **$** character.
_	CR/LF (go to beginning of a new line).

For example, to set the prompt to the current date and time, with a ">" at the end:

```
[c:\] prompt $D $t $g
Fri  Jun 6, 1997 10:29:19 >
```

To set the prompt to the current date and time, followed by the current drive and directory in upper case on the next line, with a ">" at the end:

```
[c:\] prompt $d $t$_$P$g
Fri  6-06-97 10:29:19
C:\>
```

The 4NT prompt can be set in 4START, or in any batch file that runs when 4NT starts.   The 4NT default prompt is **[$n]** (drive name in square brackets) on floppy drives, and **[$p]** (current drive and directory in square brackets) on all other drives.

If you enter PROMPT with no arguments, the prompt will be reset to its default value.   The PROMPT command sets the environment variable PROMPT, so to view the current prompt setting use the command:

```
[c:\] set prompt
```

(If the prompt is not set at all, the PROMPT environment variable will not be used, in which case the SET command above will give a "Not in environment" error.)

Along with literal text and special characters you can include the text of any environment variable, internal variable, or variable function in a prompt.   For example, if you want to include the size of the largest free memory block in the command prompt, plus the current drive and directory, you could use this command:

```
[c:\] prompt (%%@dosmem[K]K) $p$g
(601K) [c:\data]
```

Notice that the @DOSMEM function is shown with two leading percent signs [**%**].   If you used only one percent sign, the @DOSMEM function would be expanded once when the PROMPT command was executed, instead of every time the prompt is displayed.   As a result, the amount of memory would never change from the value it had when you entered the PROMPT command.   You can also use back quotes to delay expanding the variable function until the prompt is displayed:

```
[c:\] prompt `(%@dosmem[K]K) $p$g`
```

You can use this feature along with the @EXEC variable function to create a complex prompt which not only displays information but executes commands.   For example, to execute an alias which checks battery status each time the prompt is displayed (enter the alias on one line):

```
[c:\] alias cbatt `if %_apmlife lt 30 beep 440 4 880 4 440 4 880 4`
[c:\] prompt `%@exec[@cbatt]$p$g`
```

You may find it helpful to define a different prompt in secondary shells, perhaps including **$z** in the prompt to display the shell level.   To do so, place a PROMPT command in your *4START* file and use IF or IFF statements to set the appropriate prompt for different shells.

PUSHD

Purpose: Save the current disk drive and directory, optionally changing to a new drive and directory.

Format: **PUSHD [*path*]**

path: The name of the new default drive and directory.

See also: DIRS, POPD, and Directory Navigation.

Usage

PUSHD saves the current drive and directory on a "last in, first out" directory stack. The POPD command returns to the last drive and directory that was saved by PUSHD. You can use these commands together to change directories, perform some work, and return to the starting drive and directory. The DIRS command displays the contents of the directory stack.

To save the current drive and directory, without changing directories, use the PUSHD command by itself, with no *path*.

If a *path* is specified as part of the PUSHD command, the current drive and directory are saved and PUSHD changes to the specified drive and directory. If the *path* includes a drive letter, PUSHD changes to the specified directory on the new drive without changing the current directory on the original drive.

This example saves the current directory and changes to *C:\WORDP\MEMOS*, then returns to the original directory:

```
[c:\] pushd \wordp\memos
[c:\wordp\memos] popd
[c:\]
```

When you use PUSHD to change to a directory on an HPFS, NTFS, or LFN drive, you must quote the *path* name if it contains whitespace or special characters.

PUSHD can also change to a network drive and directory specified with a UNC name.

If PUSHD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database. You can also use wildcards in the *path* to force an extended directory search. Read the section on Directory Navigation for complete details.

Directory changes made with PUSHD are also recorded in the directory history list and can be displayed in the directory history window.

The directory stack can hold up to 511 characters, or between 20 to 40 typical entries (depending on the length of the names). If you exceed this limit, the oldest entry is removed before adding a new entry.

QUIT

Purpose: Terminate the current batch file.

Format: **QUIT [*value*]**

value: The numeric exit code to return to 4NT or to the previous batch file.

See also: CANCEL.

Usage

QUIT provides a simple way to exit a batch file before reaching the end of the file. If you QUIT a batch file called from another batch file, you will be returned to the previous file at the line following the original CALL.

QUIT only ends the current batch file. To end all batch file processing, use the CANCEL command.

If you specify a *value*, QUIT will set the ERRORLEVEL or exit code to that value. For information on exit codes, see the IF command and the %? variable.

You can also use QUIT to terminate an alias. If you QUIT an alias while inside a batch file, QUIT will end both the alias and the batch file and return you to the command prompt or to the calling batch file.

RD / RMDIR

Purpose: Remove one or more subdirectories.

Format: **RD [/S] *path* ...**

or

RMDIR [/S] *path* ...

path: The name of one or more subdirectories to remove.

/S(ubdirectories)

See also: MD.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

RD and RMDIR are synonyms. You can use either one.

RD removes directories from the directory tree. For example, to remove the subdirectory *MEMOS* from the subdirectory *WP*, you can use this command:

```
[c:\] rd \wp\memos
```

Before using RD, you must delete all files and subdirectories (and their files) in the *path* you want to remove. Remember to remove hidden and read-only files as well as normal files (you can use DEL /Z to delete hidden and read-only files).

You can use wildcards in the *path*.

When removing a directory on an HPFS, NTFS, or LFN drive, you must quote any *path* which contains whitespace or special characters.

If RD removes one or more directories, they will be deleted automatically from the extended directory search database.

You cannot remove the root directory, the current directory (**.**), any directory above the current directory in the directory tree, or any directory in use by another process in a multitasking system.

Options

/S (Subdirectories) This option should be used with extreme caution! It deletes all files (including hidden and system files) in the named directory and all of its subdirectories, then removes all empty subdirectories. This option can potentially erase all files on a drive with a single command.

REBOOT

Purpose: Do a system reboot.

Format: **REBOOT [/S /V]**

/L(ogoff)	**/V**(erify)
/S(hutdown)	

Usage

REBOOT will log off or shut down the operating system, or completely restart your computer. It normally performs a warm reboot, which is comparable to a shutdown and restart. The following example prompts you to verify the reboot, then does a warm boot:

```
[c:\] reboot /v
```

REBOOT defaults to performing a warm boot, with no prompting.

REBOOT flushes the disk buffers, resets the drives, and waits one second before rebooting, to allow disk caching programs to finish writing any cached data. 4NT issues the proper commands to shut down Windows NT before rebooting.

Options

/L (Logoff) Log off Windows NT, but do not reboot. This option is equivalent to the selecting Shutdown from the Start menu, then selecting "Close all programs and log on as a different user" in the shutdown dialog.

/S (Shutdown) Shut down the system, but do not reboot. This option is equivalent to selecting Shutdown from the Start menu, then selecting "Shut down the computer" in the shutdown dialog.

/V (Verify) Prompt for confirmation (**Y** or **N**) before rebooting or taking the action specified by other REBOOT options.

REM

Purpose: Put a comment in a batch file.

Format: **REM [*comment*]**

comment: The text to include in the batch file.

Usage

The REM command lets you place a remark or comment in a batch file. Batch file comments are useful for documenting the purpose of a batch file and the procedures you have used.

REM must be followed by a space or tab character and then your comment. Comments can be up to 1023 characters long. 4NT will normally ignore everything on the line after the REM command, including quote characters, redirection symbols, and other commands (see below for the exception to this rule).

If ECHO is ON, the comment is displayed. Otherwise, it is ignored. If ECHO is ON and you don't want to display the line, preface the REM command with an at sign [**@**].

You can also place a comment in a batch file by starting the comment line with two colons [**::**]. In essence this creates a batch file "label" without a valid label name. Such comments are processed slightly faster than those entered with REM, because they do not require the command processor to handle a command.

You can use REM to create a zero-byte file if you use a redirection symbol after the REM command. No text is permitted between the REM command and the redirection symbol. For example, to create the zero-byte file *C:\FOO*:

```
[c:\] rem > foo
```

(This capability is included for compatibility with *CMD.EXE*. A simpler method for creating a zero-byte file with 4NT is to use **>filename** as a command, with no actual command before the [**>**] redirection character.)

REN / RENAME

Purpose:	Rename files or subdirectories.
Format:	**REN [/A:[[-]rhsda] /E /N /P /Q /S /T] *old_name... new_name***
	or
	RENAME [/A:[[-]rhsda] /E /N /P /Q /S /T] *old_name... new_name*

old_name: Original name of the file(s) or subdirectory.
new_name: New name to use, or new path on the same drive.

/A: (Attribute select)	**/Q**(uiet)
/E (no error messages)	**/S**(ubdirectory)
/N(othing)	**/T**(otal)
/P(rompt)	

See also: COPY and MOVE.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

REN and RENAME are synonyms. You may use either one.

REN lets you change the name of a file or a subdirectory, or move one or more files to a new subdirectory on the same drive. (If you want to move files to a different drive, use MOVE.)

In its simplest form, you give REN the *old_name* of an existing file or subdirectory and then a *new_name*. The *new_name* must not already exist -- you can't give two files the same name (unless they are in different directories). The first example renames the file *MEMO.TXT* to *MEM.TXT*. The second example changes the name of the *\WORD* directory to *\WP*:

```
[c:\] rename memo.txt mem.txt
[c:\] rename \word \wp
```

If you use REN to rename a directory, the extended directory search database will be automatically updated to reflect the change.

When you rename files on an HPFS, NTFS, or LFN drive, you must quote any file names which contain whitespace or special characters. See File Names for additional details.

You can also use REN to rename a group of files that you specify with wildcards, as multiple files, or in an include list. When you do, the *new_name* must use one or more wildcards to show what part of each filename to change. Both of the next two examples change the extensions of multiple files to *.SAV*:

```
[c:\] ren config.nt autoexec.nt 4start.btm *.sav
[c:\] ren *.txt *.sav
```

REN can move files to a different subdirectory on the same drive. When it is used for this purpose, REN requires one or more filenames for the *old_name* and a directory name for the *new_name*:

```
        [c:\] ren memo.txt \wp\memos\
        [c:\] ren oct.dat nov.dat \data\save\
```

The final backslash in the last two examples is optional. If you use it, you force REN to recognize the last argument as the name of a directory, not a file. The advantage of this approach is that if you accidentally mistype the directory name, REN will report an error instead of renaming your files in a way that you didn't intend.

Finally, REN can move files to a new directory and change their name at the same time if you specify both a path and file name for *new_name*. In this example, the files are renamed with an extension of *.SAV* as they are moved to a new directory:

```
        [c:\] ren *.dat \data\save\*.sav
```

You cannot rename a subdirectory to a new location on the directory tree.

REN does not change a file's attributes. The *new_name* file(s) will have the same attributes as *old_name*.

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **REN /A: ...**), REN will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files.

/N (Nothing) Do everything except actually rename the file(s). This option is useful for testing what a REN command will actually do.

/P (Prompt) Prompt the user to confirm each rename operation. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames or the number of files renamed. This option is most often used in batch files. See also **/T**.

/S (Subdirectory) Normally, you can rename a subdirectory only if you do not use any wildcards in the *new_name*. This prevents subdirectories from being renamed inadvertently when a group of files is being renamed with wildcards. **/S** will let you rename a subdirectory even when you use wildcards. **/S** does **not** cause REN to process files in the current directory and all subdirectories as it does in some other file processing commands. To rename files throughout a directory tree, use a GLOBAL REN.

/T (Total) Don't display filenames as they are renamed, but report the number of files renamed. See also **/Q**.

RETURN

Purpose: Return from a GOSUB (subroutine) in a batch file.

Format: **RETURN [value]**

value: The exit code from 0 to 255 to return to the command processor or to the previous batch file.

See also: GOSUB.

Usage

4NT allows subroutines in batch files.

A subroutine begins with a label (a colon followed by a word) and ends with a RETURN command.

The subroutine is invoked with a GOSUB command from another part of the batch file. When a RETURN command is encountered the subroutine terminates, and execution of the batch file continues on the line following the original GOSUB. If RETURN is encountered without a GOSUB, the command processor will display a "Missing GOSUB" error.

The following batch file fragment calls a subroutine which displays the files in the current directory:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit
:subr1
dir /a/w
return
```

If you specify a ***value***, RETURN will set the ERRORLEVEL or exit code to that value. For information on exit codes see the IF command, and the **%?** variable.

SCREEN

Purpose: Position the cursor on the screen and optionally display a message.

Format: **SCREEN *row column* [*text*]**

row: The new row location for the cursor.
column: The new column location for the cursor.
text: Optional text to display at the new cursor location.

See also: ECHO, SCRPUT, TEXT, and VSCRPUT.

Usage

SCREEN allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen. You can use SCREEN to create menus, and other similar displays. The following batch file fragment displays a menu:

```
@echo off
cls
screen 3 10 Select a number from 1 to 4:
screen 6 20  1 - Word Processing         ...
```

SCREEN does not change the screen colors. To display text in specific colors, use SCRPUT or VSCRPUT. SCREEN always leaves the cursor at the end of the displayed text.

The *row* and *column* values are zero-based, so on a standard 25 line by 80 column display, valid *rows* are 0 - 24 and valid *columns* are 0 - 79. You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move the cursor down the specified number of rows or to the right the specified number of columns, or with a minus sign [**-**] to move the cursor up or to the left. This example prints a string 3 lines above the current position, in absolute column 10:

```
screen -3 10 Hello, World!
```

If you specify 999 for the *row*, SCREEN will center the text vertically on the display. If you specify 999 for the *column*, SCREEN will center the text horizontally. This example prints a message at the center of the display:

```
screen 999 999 Hello, World
```

SCREEN checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

SCRPUT

Purpose: Position text on the screen and display it in color.

Format: **SCRPUT *row col* [BRIght] *fg* ON BRIght] *bg text***

row: Starting row
col: Starting column
fg: Foreground character color
bg: Background character color
text: The text to display

See also: ECHO, SCREEN, TEXT, and VSCRPUT.

Usage

SCRPUT allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen and what colors will be used to display the message text. You can use SCRPUT to create menu displays, logos, etc.

SCRPUT works like SCREEN, but requires you to specify the display colors. See Colors and Color Names.

The *row* and *column* are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. SCRPUT checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move down the specified number of rows or to the right the specified number of columns, or with a minus sign [**-**] to move up or to the left.

If you specify 999 for the *row*, SCRPUT will center the text vertically on the display. If you specify 999 for the *column*, SCRPUT will center the text horizontally.

SCRPUT normally does not move the cursor when it displays the *text*.

The following batch file fragment displays part of a menu, in color:

```
cls white on blue
scrput 3 10 bri whi on blu Select an option:
scrput 6 20 bri red on blu 1 - Word Processing
scrput 7 20 bri yel on blu 2 - Spreadsheet
scrput 8 20 bri gre on blu 3 - Communications
scrput 9 20 bri mag on blu 4 - Quit
```

SELECT

Purpose: Interactively select files for a command.

Format: **SELECT [/A[[:][-]rhsda] /C /D /E /H /I"text" /J /L /O[[:][-]adeginrsu] /T:acw /Z] [*command*] ... (*files*...) ...**

command: The command to execute with the selected files.
files: The files from which to select. File names may be enclosed in either parentheses or square brackets. The difference is explained below.

/A(ttribute select)	**/J**(ustify names)
/C (Compression)	**/L**(ower case)
/D(isable color coding)	**/O**(rder)
/E (upper case)	**/T**(ime)
/H(ide dots)	**/X** (display short names)
/I (match descriptions)	**/Z** (use FAT format)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, or size ranges **must** appear immediately after the SELECT keyword.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

SELECT allows you to select files for internal and external commands by using a full-screen "point and shoot" display. You can have SELECT execute a command once for each file you select, or have it create a list of files for a command to work with. The *command* can be an internal command, an alias, an external command, or a batch file.

If you use parentheses around the *files*, SELECT executes the *command* once for each file you have selected. During each execution, one of the selected files is passed to the *command* as an argument. If you use square brackets around *files*, the SELECTed files are combined into a single list, separated by spaces. The command is then executed once with the entire list presented as part of its command-line arguments.

Using the SELECT File List

When you execute the SELECT command, the file list is displayed in a full-screen format which includes a top-line status bar and shows the command to be executed, the number of files marked, and the number of Kbytes in those files.

SELECT uses the cursor up, cursor down, PgUp, and PgDn keys to scroll through the file list. You can also use character matching to find specific files, just as you can in any popup window. While the file list is displayed you can enter any of the following keys to select or unselect files, display files, execute the command, or exit:

+ or **space**	Select a file, or unselect a marked file.
-	Unselect a marked file.
*****	Reverse all of the current marks (except those on subdirectories). If no files have been marked you can use ***** to mark all of the files.

/	Unselect all files.
Enter	Execute the command with the marked files, or with the currently highlighted file if no files have been marked.
Esc	Skip the files in the current display and go on to the next file specification inside the parentheses or brackets (if any)
Ctrl-C or **Ctrl-Break**	Cancel the current SELECT command entirely.

On FAT drives the file list is shown in standard FAT directory format, with names at the left an descriptions at the right. On HPFS, NTFS, and LFN drives the format is similar but more space is allowed for the name, and the description is not shown. In this format long names are truncated if they do not fit in the allowable space. For a short-name format (including descriptions) on long filename drives, use the **/X** and / or **/Z** switches.

When displaying descriptions in the short filename format, SELECT adds a right arrow at the end of the line if the description is too long to fit on the screen. This symbol will alert you to the existence of additional description text. You can use the left and right arrow keys to scroll the description area of the screen horizontally and view the additional text.

You can display the filenames in color by using the SET command to create an environment variable called COLORDIR, or using the Commands page of the OPTION dialogs or a text editor to set the ColorDir directive in your *.INI* file. If you do not use the COLORDIR variable or the ColorDir directive, SELECT will use the default screen colors for all files. See the discussion of Color-Coded Directories under DIR for more details. To disable directory color coding within SELECT, use the **/D** option.

You can set the default colors used by SELECT on the Commands page of the OPTION dialogs or with the SelectColors and SelectStatBarColors directives in the *.INI* file. If SelectColors is not used, the SELECT display will use the current default colors. If SelectStatBarColors is not used, the status bar will use the reverse of the SELECT colors.

Creating SELECT Commands

In the simplest form of SELECT, you merely specify the command and then the list of files from which you will make your selection(s). For example:

```
[c:\] select copy (*.com *.exe) a:\
```

will let you select from among the *.COM* files on the current drive, and will then invoke the COPY command to copy each file you select to drive A:. After the *.COM* files are done, the operations will be repeated for the *.EXE* files.

If you want to select from a list of all the *.COM* and *.EXE* files mixed together, create an include list inside the parentheses by inserting a semicolon:

```
[c:\] select copy (*.com;*.exe) a:\
```

Finally, if you want the SELECT command to send a single list of files to COPY, instead of invoking COPY once for each file you select, put the file names in square brackets instead of parentheses:

```
[c:\] select copy [*.com;*.exe] a:\
```

If you use brackets, you have to be sure that the resulting command (the word COPY, the list of files, and the destination drive in this example) does not exceed the command line length limit of 1,023 characters. The current line length is displayed by SELECT while you are marking files to help you to conform to this limit.

The parentheses or brackets enclosing the file name(s) can appear anywhere within the command; SELECT assumes that the first set of parentheses or brackets it finds is the one containing the list of files from which you wish to make your selection.

When you use SELECT, you must quote any file names inside the parentheses which contain whitespace or special characters. For example, to copy selected files from the *"Program Files"* directory to the *E:\SAVE* directory:

```
[c:\] select copy ("Program Files\*.*") e:\save\
```

File names passed to the *command* will be quoted automatically if they contain whitespace or special characters.

The list of files from which you wish to select can be further refined by using date, time, size, and file exclusion ranges. The range(s) must be placed immediately after the word SELECT. If the *command* is an internal command that supports ranges, an independent range can also be used in the *command* itself.

You cannot use command grouping to make SELECT execute several commands, because SELECT will assume that the parentheses are marking the list of files from which to select, and will display an error message or give incorrect results if you try to use parentheses for command grouping instead. (You **can** use a SELECT command **inside** command grouping parentheses, you just can't use command grouping to specify a group of commands for SELECT to execute.)

Advanced Topics

If you don't specify a command, the selected filename(s) will become the command. For example, this command defines an alias called UTILS that selects from the executable files in the directory *C:\UTIL*, and then executes them in the order marked:

```
[c:\] alias utils select (c:\util\*.com;*.exe;*.btm;*.bat)
```

If you want to use filename completion to enter the filenames inside the parentheses, type a space after the opening parenthesis. Otherwise, the command-line editor will treat the open parenthesis as the first character of the filename.

With the **/I** option, you can select files based on their descriptions. SELECT will display files if their description matches the text after the **/I** switch. The search is not case sensitive. You can use wildcards and extended wildcards as part of the text.

When sorting file names and extensions for the SELECT display, 4NT normally assumes that sequences of digits should be sorted numerically (for example, the file DRAW2 would come before DRAW03 because 2 is numerically smaller than 03), rather than strictly alphabetically (where DRAW2 would come second because "2" comes after "0"). You can defeat this behavior and force a strict alphabetic sort with the **/O:a** option.

Options

/A (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory

A Archive

If no attributes are listed at all (*e.g.*, SELECT **/A** ...), SELECT will display all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included in the listing. For example, **/A:RHS** will display only those files with all three attributes set.

/C (Compression) Display compression ratios on compressed NTFS drives. The compression ratio is displayed instead of the file description. The ratio is left blank for directories, for files with a length of 0 bytes, and for files on non-compressed drives. The compression ratios will not be visible on LFN or NTFS drives unless you use **/Z** to switch to the traditional short filename format.

See **DIR** **/C** for more details on how compression ratios are calculated.

/D (Disable color coding) Temporarily turn off directory color coding within SELECT.

/E (use upper case) Display filenames in upper case; also see SETDOS /U and the UpperCase directive in the *4NT.INI* file.

/H (Hide dots) Suppress the display of the "." and ".." directories.

/I (match descriptions) Display filenames by matching text in their descriptions. The text can include wildcards and extended wildcards. The search text must be enclosed in quotation marks. You can select all filenames that have a description with **/I"[?]*"**, or all filenames that do not have a description with **/I"[]"**. **/I** will be ignored if **/C** or **/O:c** is also used.

/L (Lower case) Display file and directory names in lower case; also see SETDOS /U and the UpperCase directive in *4NT.INI*.

/J (Justify names) Justify (align) filename extensions and display them in the traditional format.

/O (Order) Set the sort order for the files. The order can be any combination of the following options:

- Reverse the sort order for the next option.
a Sort in ASCII order, not numerically, when there are digits in the name
c Sort by compression ratio (the least compressed file in the list will be displayed first). For information on supported compression systems see **/C** above.
d Sort by date and time (oldest first).
e Sort by extension.
g Group subdirectories first, then files.
i Sort by file description.
n Sort by filename (this is the default).
r Reverse the sort order for all options.
s Sort by size.
u Unsorted.

/T:acw (Time display) Specify which of the date and time fields on an LFN, NTFS, or HPFS drive should be displayed and used for sorting:

a last access date and time (access time is not saved on LFN volumes).
c creation date and time.
w last write date and time (default).

/X Display short filenames, in the traditional FAT format (like **/Z**), on NTFS and LFN drives.

/Z Display files on an HPFS, NTFS, or LFN drive in the traditional FAT format, with the filename at the left and the description at the right. On LFN and NTFS drives, short filenames will be displayed. On HPFS drives, long names will be truncated to 12 characters; if the name is longer than 12 characters, it will be followed by a right arrow to show that one or more characters have been truncated.

SET

Purpose: Display, create, modify, or delete environment variables.

Format: **SET [/A /P /R *file*...] [*name*[=][*value*]]**

file: One or more files containing variable definitions.
name: The name of the environment variable to define or modify.
value: The new value for the variable.

/A(rithmetic) **/R**(ead from file)
/P(ause)

See also: ESET and UNSET.

Usage

Every program and command inherits an environment, which is a list of variable *names*, each of which is followed by an equal sign and some text. Many programs use entries in the environment to modify their own actions.

If you simply type the SET command with no options or arguments, it will display all the names and values currently stored in the environment. Typically, you will see an entry called COMSPEC, an entry called PATH, an entry called CMDLINE, and whatever other environment variables you and your programs have established:

```
[c:\] set
COMSPEC=C:\4NT\$OS2.EXE
PATH=C:\;C:\WINDOWS;C:\WINDOWS\SYSTEM;C:\UTIL
CMDLINE=C:\4NT\4START.CMD
```

To add a variable to the environment, type SET, a space, the variable name, an equal sign, and the text:

```
[c:\] set mine=c:\finance\myfiles
```

The variable name is converted to upper case by 4NT. The text after the equal sign will be left just as you entered it. If the variable already exists, its value will be replaced with the new text that you entered.

Normally you should not put a space on either side of the equal sign. A space before the equal sign will become part of the *name* ; a space after the equal sign will become part of the *value*.

If you use SET to create a variable with the same name as one of the 4NT internal variables, you will disable the internal variable. If you later execute a batch file or alias that depends on that internal variable, it may not operate correctly.

To display the contents of a single variable, type SET plus the variable name:

```
[c:\] set mine
```

You can edit environment variables with the ESET command. To remove variables from the environment, use UNSET, or type SET plus a variable name and an equal sign:

```
[c:\] set mine=
```

The variable *name* is limited to a maximum of 80 characters. The name and *value* together cannot be longer than 1,023 characters.

In 4NT the size of the environment is set automatically, and increased as necessary as you add variables.

Options

/A (Arithmetic) Evalute the argument to the right of the equal sign, place the result in the environment, and display it. You can use @EVAL to perform the same task; SET /A is included only for compatibility with Windows NT's *CMD.EXE*. The following example adds 2 and 2, and places 4 in the environment variable VAR:

```
[c:\] set /a var=2+2
```

In addition, **/A** interprets alphabetic strings to the right of the equal sign as environment variable names even if they are not preceded by a percent sign. For example, this sequence will set **Y** to 4:

```
[c:\] set x=2
[c:\] set /a y=x+2
```

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/R (Read) Read environment variables from a file. This is much faster than loading variables from a batch file with multiple SET commands. Each entry in the file must fit within the 1,023-byte command-line length limit for 4NT: The file is in the same format as the SET display (*i.e.*, **name=value**), so SET /R can accept as input a file generated by redirecting SET output. For example, the following commands will save the environment variables to a file, and then reload them from that file:

```
set > varlist
set /r varlist
```

You can load variables from multiple files by listing the filenames individually after the **/R**. You can add comments to a variable file by starting the comment line with a colon [**:**].

If you are creating a SET /R file by hand, and need to create an entry that spans multiple lines in the file, you can do so by terminating each line, except the last, with an escape character. However, you cannot use this method to exceed the command-line length limit.

SETDOS

Purpose: Display or set the 4NT configuration.

Format: **SETDOS [/C? /D? /E? /Fn.n /G?? /I+|- command /M? /N? /P? /S?:? /U? /V? /X[+|-]n /Y?]**

/C(ompound)	**/N**(o clobber)
/D(escriptions)	**/P**(arameter character)
/E(scape character)	**/S**(hape of cursor)
/F(ormat for @EVAL)	**/U**(pper case)
/G (numeric separators)	**/V**(erbose)
/I(nternal commands)	**/X** (expansion, special characters)
/M(ode for editing)	**/Y** (debug batch file)

Usage

SETDOS allows you to customize certain aspects of 4NT to suit your personal tastes or the configuration of your system. Each of these options is described below.

You can display the value of all SETDOS options by entering the SETDOS command with no parameters.

Most of the SETDOS options can be initialized when 4NT executes the configuration directives in the *.INI* file, and can also be set on the Command Line 1, Command Line 2, Options 1, or Options 2 page of the OPTION dialogs. The name of the corresponding directive is listed with each option below; if none is listed, that option cannot be set with OPTION or from the *.INI* file. You can also define the SETDOS options in your *4START* or other startup file (see Automatic Batch Files), in aliases, or at the command line.

Secondary shells automatically inherit most configuration settings currently in effect in the previous shell. If values have been changed by SETDOS since 4NT started, the new values will be passed to the secondary shell.

SETDOS /I settings are not inherited by secondary shells. If you want to use SETDOS /I- to disable commands in all shells, place the SETDOS command(s) in your *4START* file, which is executed when any shell starts.

Options

/C (Compound character) This option sets the character used for separating multiple commands on the same line. The default is the ampersand [**&**]. You cannot use any of the redirection characters (**| > <**), or the blank, tab, comma, or equal sign as the command separator. The command separator is saved by SETLOCAL and restored by ENDLOCAL. This example changes the separator to a tilde [**~**]:

```
[c:\] setdos /c~
```

If you want to share batch files or aliases between 4NT and 4DOS, 4OS2, or Take Command, see the %+ variable, which retrieves the current command separator, and Special Character Compatibility for details on using compatible command separators for all the products you use.

Also see the CommandSep directive.

/D (Descriptions) This option controls whether file processing commands like COPY, DEL, MOVE, and REN process file descriptions along with the files they belong to. **/D1** turns description processing on, which is the default. **/D0** turns description processing off. Also see the Descriptions directive.

You can also use **/D** to set the name of the hidden file in each directory that contains file descriptions. To do so, follow **/D** with the filename in quotes:

```
   [c:\] setdos /d"files.bbs"
```

Use this option with caution, because changing the name from the default will make it difficult to transfer file descriptions to another system. This option is provided for bulletin board system operators and others who have special needs. Also see the DescriptionName directive.

/E (Escape character) This option sets the character used to suppress the normal meaning of the following character. Any character following the escape character will be passed unmodified to the command. The default escape character is a caret [**^**]. You cannot use any of the redirection characters (**| > <**) or the blank, tab, comma, or equal sign as the escape character. The escape character is saved by SETLOCAL and restored by ENDLOCAL. Certain characters (**b**, **c**, **e**, **f**, **k**, **n**, **q**, **r**, **s**, and **t**) have special meanings when immediately preceded by the escape character.

If you want to share batch files or aliases between 4NT and 4DOS, 4OS2, or Take Command, see the %= variable, which retrieves the current escape character, and Special Character Compatibility for details on using compatible escape characters for all the products you use.

Also see the EscapeChar directive.

/F (Format for @EVAL) This option lets you set default decimal precision for the @EVAL variable function. The maximum precision is 16 digits to the left of the decimal point and up to 8 digits to the right of the decimal point.

The general form of this option is **/Fx.y**, where the x value sets the minimum number of digits to the right of the decimal place and the y value sets the maximum number of digits. You can use **=x,y** instead of **=x.y** if the comma is your decimal separator. Both values can range from 0 to 8; if **x** is greater than **y**, it is ignored. You can specify either or both values: /F2.5, /F2, and /F.5 are all valid entries. See the @EVAL function if you want to set the precision for a single computation.

Also see the EvalMax and EvalMin directives.

/G (Numeric separators) This option sets the decimal and thousands separator characters. The format is **/Gxy** where "x" is the new decimal separator and "y" is the new thousands separator. Both characters must be included. The only valid settings are **/G.,** (period is the decimal separator, comma is the thousands separator); **/G,.** (the reverse); or **/G0** to remove any custom setting and use the default separators associated with your current country code (this is the default).

The decimal separator is used for @EVAL, numeric IF and IFF tests, version numbers, and other similar uses. The thousands separator is used for numeric output, and is skipped when performing calculations in @EVAL.

Also see the DecimalChar and ThousandsChar directives.

/I (Internal) This option allows you to disable or enable internal commands. To disable a command, precede the command name with a minus [**-**]. To re-enable a command, precede it with a plus [**+**]. For example, to disable the internal LIST command to force 4NT to use an external command:

```
   [c:\] setdos /i-list
```

/M (Mode) This option controls the initial line editing mode. To start in overstrike mode at the beginning of each command line, use **/M0** (the default). To start in insert mode, use **/M1**. Also see the EditMode directive.

/N (No clobber) This option controls output redirection). **/N0** means existing files will be overwritten by output redirection (with **>**) and that appending (with **>>**) does not require the file to exist already. This is the default. **/N1** means existing files may not be overwritten by output redirection, and that when appending the output file must exist. A **/N1** setting can be overridden with the [**!**] character. If you use **/N1**, you may have problems with a few unusual programs that shell out to run a command with redirection, and expect to be able to overwrite an existing file.

Also see the NoClobber directive.

/P (Parameter character) This option sets the character used after a percent sign to specify all or all remaining command-line arguments in a batch file or alias (*e.g.*, **%&** or **%n&**). The default is the dollar sign [**$**]. The parameter character is saved by SETLOCAL and restored by ENDLOCAL.

If you want to share batch files or aliases between 4NT and 4DOS, 4OS2, or Take Command, see Special Character Compatibility for details on selecting compatible parameter characters for all the products you use.

Also see the ParameterChar directive.

/S (Shape) This option sets the cursor shape. The format is **/So:i** where **o** is the cursor size for overstrike mode, **i** the cursor size for insert mode. The size is entered as a percentage of the total character height. The default values are 10:100 (a 10% underscore cursor for overstrike mode, and a 100% block cursor for insert mode). Because of the way video drivers remap the cursor shape, you may not get a smooth progression in the cursor size from 0% - 100%. To disable the cursor, enter **/S0:0**.

If either value is -1, the command processor will not attempt to modify the cursor shape at all. You can use this feature to give another program full control of the cursor shape. You can retrieve the current cursor shape values with the **%_CI** and **%_CO** internal variables

Also see the CursorOver and CursorIns directives.

/U (Upper) This option controls the default case (upper or lower) for file and directory names displayed by internal commands like COPY and DIR. **/U0** displays file names in lower case (the default). **/U1** displays file names in the traditional upper case. Also see the UpperCase directive.

The **/U** setting is ignored for filenames on LFN, HPFS, and NTFS drives. Names on such drives are always displayed in the case in which they are stored.

/V (Verbose) This option controls the default for command echoing in batch files. **/V0** disables echoing of batch file commands unless ECHO is explicitly set ON. **/V1**, the default setting, enables echoing of batch file commands unless ECHO is explicitly set OFF. Also see the BatchEcho directive.

/V2 forces echoing of all batch file commands, even if ECHO is set OFF or the line begins with an "@". This allows you to turn echoing on for a batch file without editing the batch file and removing the ECHO OFF command(s) within it. **/V2** is intended for debugging, and can be set with SETDOS, but not with the OPTION command or the BatchEcho directive in *4NT.INI*.

/X[+|-]n (expansion and special characters) This option enables and disables alias and environment variable expansion, and controls whether special characters have their usual meaning or are treated as text. It is most often used in batch files to process text strings which may contain special characters.

The features enabled or disabled by **/X** are numbered. All features are enabled when 4NT starts, and you can re-enable all features at any time by using **/X0**. To disable a particular feature, use **/X-n**, where **n** is the feature number from the list below. To re-enable the feature, use **/X+n**. To enable or disable multiple individual features, list their numbers in sequence after the **+** or **-** (*e.g.* **/X- 345** to disable features 3, 4, and 5).

The features are:

1	All alias expansion
2	Nested alias expansion only
3	All variable expansion (environment variables and batch and alias parameters)
4	Nested variable expansion only
5	Multiple commands, conditional commands, and piping
6	Redirection
7	Quoting (double quotes and back quotes) and square brackets
8	Escape character

If nested alias expansion is disabled, the first alias of a command is expanded but any aliases it invokes are not expanded. If nested variable expansion is disabled, each variable is expanded once, but variables containing the names of other variables are not expanded further.

For example, to disable all features except alias expansion while you are processing a text file containing special characters:

```
setdos /x-35678
... [perform text processing here]
setdos /x0
```

/Y (Debug batch file) **/Y1** enables the built-in batch file debugger. The debuggger allows you to "single-step" through a batch file line by line, with the file displayed in a popup window as it executes. For complete details on using the debugger see Debugging Batch Files (this topic also covers additional debugging techniques which do not require stepping through each line individually).

To start the debugger, insert a SETDOS /Y1 command at the beginning of the portion of the batch file you want to debug, and a SETDOS /Y0 command at the end.

You cannot use the batch debugger with REXX files or EXTPROC files. It can only be used with normal 4NT batch files.

You can also invoke SETDOS /Y1 from the prompt, but because the debugger is automatically turned off whenever the command processor returns to the prompt, you must enter the SETDOS command and the batch file name on the same line, for example:

```
[c:\] setdos /y1 & mybatch.btm
```

SETLOCAL

Purpose: Save a copy of the current disk drive, directory, environment, alias list, and special characters.

Format: **SETLOCAL**

See also: ENDLOCAL.

Usage

SETLOCAL is used in batch files to save the default disk drive and directory, the environment, the alias list, and the command separator, escape character, parameter character, decimal separator, and thousands separator. You can then change their values and later restore the original values with ENDLOCAL.

For example, this batch file fragment saves everything, removes all aliases so that user aliases will not affect batch file commands, changes the disk and directory, changes the command separator, runs a program, and then restores the original values:

```
setlocal
unalias *
cdd d:\test
setdos /c~
program ~ echo Done!
endlocal
```

SETLOCAL and ENDLOCAL are not nestable within a batch file. However, you can have multiple, separate SETLOCAL / ENDLOCAL pairs within a batch file, and nested batch files can each have their own SETLOCAL / ENDLOCAL. You cannot use SETLOCAL in an alias or at the command line.

An ENDLOCAL is performed automatically at the end of a batch file if you forget to do so. If you invoke one batch file from another without using CALL, the first batch file is terminated, and an automatic ENDLOCAL is performed; the second batch file inherits the settings as they were prior to any SETLOCAL.

SHIFT

Purpose: Allows the use of more than 127 parameters in a batch file.

Format: **SHIFT [*n* | /n]**

n: Number of positions to shift.

Usage

SHIFT is provided for compatibility with older batch files, where it was used to access more than 10 parameters. 4NT supports 128 parameters (%0 to %127), so you may not need to use SHIFT for batch files running exclusively under JP Software command processors.

SHIFT moves each of the batch file parameters *n* positions to the left. The default value for *n* is 1. SHIFT 1 moves the parameter in %1 to position %0, the parameter in %2 becomes %1, etc. You can reverse a SHIFT by giving a negative value for *n* (*i.e.*, after SHIFT -1, the former %0 is restored, %0 becomes %1, %1 becomes %2, etc.).

SHIFT also affects the parameters **%n$** (command-line tail) and **%#** (number of command arguments).

If you add a slash before the value ***n***, the value determines the postion at which to begin the shift. For example:

```
        shift /2
```

leaves parameters %0 and %1 unchanged, and moves the value of %3 to postion %2, %4 to %3, etc. The value after the slash cannot be negative, and shifts performed with the slash cannot be undone later in the batch file.

SHRALIAS

Purpose: Retains global command history, directory history, and alias lists in memory when the command processor is not running.

Format: **SHRALIAS [/U]**

/U(nload)

See also: ALIAS, command history and recall, and directory history window.

Usage

When you close all 4NT sessions, the memory for the global command history, global directory history, and global alias list is released. If you want the lists to be retained in memory even when no command processor session is running, you need to execute SHRALIAS.

The SHRALIAS command starts and initializes *SHRALIAS.EXE*, a small program which remains active and retains global lists when 4NT is not running. In order to start the program, SHRALIAS must be able to find *SHRALIAS.EXE* either in the same directory as 4NT, or in a directory in your PATH. You cannot run *SHRALIAS.EXE* directly, it must be run by the SHRALIAS command.

Once SHRALIAS has been executed, the global lists will be retained in memory until you use **SHRALIAS /U** to unload the lists, or until you shut down your operating system.

SHRALIAS will not work unless you have at least one copy of 4NT running with global alias, command history, and directory history lists enabled. If the required global lists are not found, SHRALIAS will display an error.

If you start SHRALIAS from a temporary 4NT session which exits after starting SHRALIAS (for example, by executing SHRALIAS in your *STARTUP.CMD* file), the command processor session may terminate and discard the shared lists before SHRALIAS can attach to them. In this case *SHRALIAS.EXE* will not be loaded. If you experience this problem, add a short delay with the DELAY command after SHRALIAS is loaded and before your session exits.

SHRALIAS does not work properly in detached sessions (*e.g.* those started with DETACH, or with Windows NT's AT utility), due to security issues within Windows NT. The SHRALIAS command will be ignored in detached sessions.

For more information about global history and alias lists, see Local and Global Command History, Local and Global Directory History, and local and global alias lists.

Options

/U (Unload) Shuts down *SHRALIAS.EXE*. If SHRALIAS is not loaded again, the memory used by global command history, directory history, and alias lists will be released when the last copy of 4NT exits.

START

Purpose: Start a program in another session or window.

Format: **START ["*program title* "] [/B /C /D*path*] /HIGH /I /INV /K /L /LA /LD /LH /LOW /MAX /MIN /N /NORMAL /PGM *progname* /POS=x, y, width, height /REALTIME /SEPARATE /SHARED /SIZE=rows, cols /WAIT] [*command*]**

program title: Title to appear on title bar.
progname: Program name (not the session name).
path: Startup directory.
command: Command to be executed.

/B (no new console)	**/MAX**(imized)
/C(lose when done)	**/MIN**(imized)
/D(irectory)	**/N**(o *command processor*)
/HIGH (priority)	**/NORMAL** (priority)
/I(nherit environment)	**/PGM** (program name)
/INV(isible)	**/POS**(ition of window)
/K(eep when done)	**/REALTIME** (priority)
/L(ocal lists)	**/SEPARATE** (virtual machine)
/LA (local aliases)	**/SHARED** (WoW VDM)
/LD (local directory history)	**/SIZE** (of screen buffer)
/LH (local history list)	**/WAIT** (for session to finish)

See also: DETACH.

Usage

START is used to begin a new Windows NT session, and optionally run a program in that session. If you use START with no parameters, it will begin a new command-line session. If you add a *command*, START will begin a new session or window and execute that command. START will determine the application type automatically and start the session in the appropriate mode.

The *program title*, if it is included, will appear on the task list and Alt-Tab displays. The *program title* must be enclosed in quotation marks and cannot exceed 127 characters. If the *program title* is omitted, the program name will be used as the title.

START always assumes that the first quoted string on the command line is the *program title*; if there is a second quoted string it is assumed to be the *command*. As a result, if the name of the program you are starting is a long filename containing whitespace (and must therefore be quoted), you cannot simply place it on the command line. If you do, as the first quoted string it will be interpreted as the *program title*, not the *command*. To address this, use the **/PGM** switch to indicate explicitly that the quoted string is the program name, or include a title before the program name. For example, to start the program "*C:\Program Files\Proc.Exe*" you could use either of the first two commands below, but the third command would not work:

```
[c:\] start /PGM "C:\Program Files\Proc.Exe"
[c:\] start "test" "C:\Program Files\Proc.Exe"
[c:\] start "C:\Program Files\Proc.Exe"
```

If the *progname* is the name of a directory instead of an executable program, 4NT will start Windows Explorer in the specified directory if you are using Windows NT 4.0 or later. (Explorer must be in the PATH, the *\WINDOWS* directory, or the *\WINDOWS\SYSTEM* directory for this feature to work correctly.)

START offers a large number of switches to control the session you start. In most cases you need only a few switches to accomplish what you want. The list below summarizes the most commonly used START options, and how you can use them to control the way a session is started:

/MAX, **/MIN**, and **/POS** allow you to start a character-mode windowed session in a maximized window, a minimized window, or a window with a specified position and size. The default is to let the operating environment choose the position and size of the window.

/C allows you to close the session when the command is finished (the default for Windows NT Presentation graphical sessions); **/K** allows you to keep the session open and go to a prompt (the default for Windows NT character mode sessions).

Options

/B	(No new console) The program is started without creating a new window or console, *i.e.* in the 4NT window. Normally, the application is started in its own window. For compatibility with *CMD.EXE*, **/B** also disables Ctrl-C processing for the program.
/C	(Close) The session or window is closed when the application ends.
/D	(Directory) Specifies the startup directory. Include the directory name immediately after the **/D**, with no intervening spaces or punctuation.
/HIGH	Start the window at high priority.
/I	(Inherit environment) Inherit the default environment, if any, rather than the current environment.
/INV	(Invisible) Start the session or window as invisible. No icon will appear and the session will only be accessible through the Task Manager or Window List.
/K	(Keep session or window at end) The session or window continues after the application program ends. Use the EXIT command to end the session.
/L	(Local lists) Start 4NT with local alias, history, and directory history lists. This option combines the effects of **/LA, /LD,** and **/LH** (below).
/LA	(Local Alias list) Start 4NT with a local alias list. See ALIAS for information on local and global aliases.
/LD	(Local Directory history list) Start 4NT with a local directory history list. See Directory History Window for information on local and global directory history.
/LH	(Local History list) Start 4NT with a local history list. See Command History and Recall for information on local and global history lists.
/LOW	Start the window at low priority.
/MAX	(Maximized) Start the session or window maximized.
/MIN	(Minimized) Start the session or window minimized.
/N	Don't invoke *4NT.EXE* to run the command.
/NORMAL	Start the window at normal priority.

/PGM (Program name) The string following this option is the program name. If you do not use **/PGM**, the first quoted string on the line will be used as the session and task list title, and not as the program name.

/POS (Position) Start the window at the specified screen position. The syntax is **/POS=x, y, width, height** where the values are specified in pixels or pels. **x** and **y** refer to the position of the bottom left corner of the window relative to the bottom left corner of the screen.

/REALTIME Start the window at realtime priority.

/SEPARATE Start a 16-bit Windows application in a separate virtual machine. Normally, all 16-bit Windows applications are started in the same virtual machine.

/SHARED Start a 16-bit Windows application in the shared virtual machine (the opposite of /SEPARATE). This is the default; the switch is included only for compatibility with *CMD.EXE*.

/SIZE Start the window with the specified screen buffer size. The full syntax is **/SIZE=rows, columns**, where **rows** is the number of text rows and **columns** is the number of text columns.

/WAIT Wait for the new session or window to finish before continuing.

SWITCH

Purpose: Select commands to execute based on a value.

Format: **SWITCH expression**

CASE value1 [.OR. value2] ...

commands

CASE value3

commands

[DEFAULT

commands]

ENDSWITCH

expression: An environment variable, internal variable, variable function, text string, or a combination of these elements, that is used to select a group of commands.
***value1**, **value2**, etc.*: A value to test, or multiple values connected with **.OR.**
commands: One or more commands to execute if the expression matches the *value*. If you use multiple commands, they must be separated by command separators or placed on separate lines.

See also: IF and IFF.

Usage

SWITCH can only be used in batch files. It allows you to select a command or group of commands to execute based on the possible values of a variable or a combination of variables and text.

The SWITCH command is always followed by an *expression* created from environment variables, internal variables, variable functions, and text strings, and then by a sequence of CASE statements matching the possible *values* of the *expression*. If one of the *values* in a CASE statement matches the *expression*, the commands following that CASE statement are executed, and all subsequent CASE statements and the commands which follow them are ignored. If no matches are found, the commands following the optional DEFAULT statement are executed. If there are no matches and there is no DEFAULT statement, no commands are executed by SWITCH.

After all of the commands following the CASE or DEFAULT statement are executed, the batch file continues with the commands that follow ENDSWITCH.

You must include a command separator or new line after the *expression*, before each CASE or DEFAULT statement, before each command, and before ENDSWITCH. You can link *values* in a CASE statement with **.OR.** (but not with **.AND.** or **.XOR.**).

For example, the following batch file fragment displays one message if the user presses **A**, another if user presses **B** or **C**, and a third if the user presses any other key:

```
inkey Enter a keystroke: %%key
switch %key
case A
```

```
   echo It's an A
case B .or. C
   echo It's either B or C
default
   echo It's not A, B, or C
endswitch
```

In the example above, the value of a single environment variable was used for the *expression*. You will probably find that this is the best method to use in most situations. However, you can use other kinds of expressions if necessary. The first example below selects a command to execute based on the length of a variable, and the second bases the action on a quoted text string stored in an environment variable:

```
switch %@len[%var1]
case 0
   echo Missing var1
case 1
   echo Single character
...
endswitch
switch "%string1"
case "This is a test"
   echo Test string
case "The quick brown fox"
   echo It's the fox
...
endswitch
```

The SWITCH and ENDSWITCH commands must be on separate lines, and cannot be placed within a command groupor on the same line as other commands (this is the reason SWITCH cannot be used in aliases). However, commands within the SWITCH block can use command groups or the command separator in the normal way.

SWITCH commands can be nested.

You can exit from all SWITCH / ENDSWITCH processing by using GOTO to a line past the last ENDSWITCH.

TEE

Purpose: Copy standard input to both standard output and a file.

Format: **TEE [/A] *file*...**

file: One or more files that will receive the "tee-d" output.

/A(ppend)

See also: Y and the redirection options.

Usage

TEE is normally used to "split" the output of a program so that you can see it on the display and also save it in a file. It can also be used to capture intermediate output before the data is altered by another program or command.

TEE gets its input from standard input (usually the piped output of another command or program), and sends out two copies: one goes to standard output, the other to the *file* or *files* that you specify. TEE is not likely to be useful with programs which do not use standard output, because these programs cannot send output through a pipe.

For example, to search the file *DOC* for any lines containing the string "4NT", make a copy of the matching lines in *4.DAT*, sort the lines, and write them to the output file *4N.DAT*:

```
[c:\] find "4NT" doc | tee 4.dat | sort > 4n.dat
```

If you are typing at the keyboard to produce the input for TEE, you must enter a **Ctrl-Z** to terminate the input.

When using TEE with a pipe under 4NT, the programs on the two ends of the pipe run simultaneously, not sequentially as in 4DOS.

See Piping for more information on pipes.

Options

/A (Append) Append the output to the file(s) rather than overwriting them.

TEXT

Purpose: Display a block of text in a batch file.

Format: **TEXT**

```
.
.
.
```

ENDTEXT

See also: ECHO, SCREEN, SCRPUT, and VSCRPUT.

Usage

TEXT can only be used in batch files.

The TEXT command is useful for displaying menus or multi-line messages. TEXT will display all subsequent lines in the batch file until terminated by ENDTEXT. Both TEXT and ENDTEXT must be entered as the only command on the line.

To redirect the entire block of text, use redirection on the TEXT command itself, but not on the actual text lines or the ENDTEXT line. No environment variable expansion or other processing is performed on the lines between TEXT and ENDTEXT; they are displayed exactly as they are stored in the batch file.

You can use a CLS or COLOR command to set the screen color before executing the TEXT command.

The following batch file fragment displays a simple menu:

```
@echo off & cls
screen 2 0
text
Enter one of the following:
1 - Spreadsheet
2 - Word Processing
3 - Utilities
4 - Exit
endtext
inkey /k"1234" Enter your selection:  %%key
```

TIME

Purpose: Display or set the current system time.

Format: **TIME [/T] [*hh*[:mm[:ss]]] [AM | PM]**

hh: The hour (0 - 23).
mm: The minute (0 - 59).
ss: The second (0 - 59).

/T (display only)

See also: DATE.

Usage

If you don't enter any parameters, TIME will display the current system time and prompt you for a new time. Press **Enter** if you don't wish to change the time; otherwise, enter the new time:.

```
[c:\] time
Mon  Dec 22, 1997  9:30:06
New time (hh:mm:ss):
```

TIME defaults to 24-hour format, but you can optionally enter the time in 12-hour format by appending "a", "am", "p", or "pm" to the time you enter.

For example, to enter the time as 9:30 am:

```
[c:\] time 9:30 am
```

Windows NT adds the system time and date to the directory entry for every file you create or modify. If you keep both the time and date accurate, you will have a record of when you last updated each file.

Options

/T Displays the current time but does not prompt you for a new time. You cannot specify a new time on the command line with **/T**. If you do, the new time will be ignored.

TIMER

Purpose: TIMER is a system stopwatch.

Format: **TIMER [ON] [/1 /2 /3 /S]**

ON: Force the stopwatch to restart

/1 (stopwatch #1)	**/3** (stopwatch #3)
/2 (stopwatch #2)	**/S**(plit)

Usage

The TIMER command turns a system stopwatch on and off. When you first run TIMER, the stopwatch starts:

```
[c:\] timer
Timer 1 on:  12:21:46
```

When you run TIMER again, the stopwatch stops and the elapsed time is displayed:

```
[c:\] timer
Timer 1 off:  12:21:58
Elapsed time: 0:00:12.06
```

There are three stopwatches available (1, 2, and 3) so you can time multiple overlapping events. By default, TIMER uses stopwatch #1.

TIMER is particularly useful for timing events in batch files. For example, to time both an entire batch file, and an intermediate section of the same file, you could use commands like this:

```
rem Turn on timer 1
timer
rem Do some work here
rem Turn timer 2 on to time the next section
timer /2
rem Do some more work
echo Intermediate section completed
rem Display time taken in intermediate section
timer /2
rem Do some more work
rem Now display the total time
timer
```

The smallest interval TIMER can measure depends on the operating system you are using, your hardware, and the interaction between the two. However, it should never be greater than .06 second. The largest interval is 23 hours, 59 minutes, 59.99 seconds.

Options

/1 Use timer #1 (the default).

/2 Use timer #2.

/3 Use timer #3.

/S (Split) Display a split time without stopping the timer. To display the current elapsed time but leave the timer running:

```
[c:\] timer /s
Timer 1 elapsed: 0:06:40.63
```

ON Start the timer regardless of its previous state (on or off). Otherwise the TIMER command toggles the timer state (unless **/S** is used).

TITLE

Purpose: Change the window title.

Format: **TITLE "*title*"**

title: The new window title.

See also: ACTIVATE and WINDOW.

Usage

TITLE changes the text that appears in the caption bar at the top of the 4NT window. You can also change the window title with the WINDOW command or the ACTIVATE command.

The title text should not be enclosed in quotes unless you want the quotes to appear as part of the actual title.

To change the title of the current window to "JP Software / 4NT":

```
        [c:\] title JP Software / 4NT
```

TOUCH

Purpose: Change a file's date and time stamps.

Format: **TOUCH [/C /D[acw][mm-dd-yy] /E /F /Q /T[acw][hh:mm]** ***file...***

file: One or more files whose date and/or time stamps are to be changed.

/C(reate file)	**/F**(orce read-only files)
/D(ate)	**/Q**(uiet)
/E (No error messages)	**/T**(ime)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Use extended wildcards with caution on LFN volumes; see LFN File Searches for details.

Usage

TOUCH is used to change the date and / or time of a file. You can use it to be sure that particular files are included or excluded from an internal command, backup program, compiler MAKE utility, or other program that selects files based on their time and date stamps, or to set a group of files to the same date and time for consistency.

TOUCH should be used with caution, and in most cases should only be used on files you create. Many programs depend on file dates and times to perform their work properly. In addition, many software manufacturers use file dates and times to signify version numbers. Indiscriminate changes to date and time stamps can lead to confusion or incorrect behavior of other software.

TOUCH normally works with existing files, and will display an error if the *file* you specify does not exist, or has the read-only attribute set. To create the *file* if it does not already exist, use the **/C** switch. To force a date and time change for read-only files, use the **/F** switch.

TOUCH displays the date, time, and full name of each file whose timestamp is modified. To disable this output, use **/Q**.

If you don't specify a date or a time, TOUCH will default to the current date and time from your system clock. For example, to set the time stamp of all *.C* files in the current directory to the current date and time:

```
[d:\source] touch *.c
 6-12-97 11:13:58  D:\SOURCE\MAIN.C
 6-12-97 11:13:58  D:\SOURCE\INIT.C
 ...
```

If you specify a date but not a time, the time will default to the current time from your system clock. Similarly, if you specify a time but not a date, the date will be obtained from the system clock.

On LFN, HPFS, and NTFS files, TOUCH sets the "modified" or "last write" date and time by default. By adding the appropriate character to the **/D** or **/T** switch, you can set the other date and time stamps that are maintained for each file:

a	last access date and time (access time can not be set on LFN volumes).

c	creation date and time.
w	last write date and time (default).

Options

/C	(Create file) Create the *file* (as a zero-byte file) if it does not already exist. You cannot use wildcards with **/C**, but you can create multiple *files* by listing them individually on the command line.
/D	(Date) Specify the date that will be set for the selected files. If the date is not specified, TOUCH will use the current date. For LFN, HPFS, and NTFS files you can use **/Da**, **/Dc**, or **/Dw**, followed by the date, to explicitly specify the last access, creation, or last write date stamp. The date must be entered using the proper format for your current country settings.
/E	(No error messages) Suppress all non-fatal error messages, such as "File not found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files.
/F	(Force read-only files) Remove the read-only attribute from each file before changing the date and time, and restore it afterwards. Without **/F**, attempting to change the date and time on a read-only file will usually cause an error.
/Q	(Quiet) Do not display the new date and time and the full name for each file.
/T	(Time) Specify the time that will be set for the selected files, in hh:mm format. If the time is not specified, TOUCH will use the current time. For LFN, HPFS, and NTFS files you can use **/Ta**, **/Tc**, or **/Tw**, followed by the time, to explicitly specify the last access, creation, or last write time stamp. However, files on LFN volumes do not have a last access time, so **TOUCH /Ta** will have no effect on such files.

TREE

Purpose: Display a graphical directory tree.

Format: **TREE [/A /B /F /H /P /S /T[:acw]]** ***dir...***

dir: The directory to use as the start of the tree. If more than one directory is specified, TREE will display a directory tree for each.

/A(SCII)	**/P**(ause)
/B(are)	**/S** (file size)
/F(iles)	**/T**(ime and date)
/H(idden directories)	

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

The TREE command displays a graphical representation of the directory tree using standard or extended ASCII characters. For example, to display the directory structure on drive C:

```
[c:\] tree c:\
```

You can print the display, save it in a file, or view it with LIST by using standard redirection symbols. Be sure to review the **/A** option before attempting to print the TREE output. The options, discussed below, specify the amount of information included in the display.

Options

/A (ASCII) Display the tree using standard ASCII characters. You can use this option if you want to save the directory tree in a file for further processing or print the tree on a printer which does not support the graphical symbols that TREE normally uses.

/B (Bare) Display the full pathname of each directory, without any of the line-drawing characters.

/F (Files) Display files as well as directories. If you use this option, the name of each file is displayed beneath the name of the directory in which it resides.

/H (Hidden) Display hidden as well as normal directories. If you combine **/H** and **/F**, hidden files are also displayed.

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/S (Size) Display the size of each file. This option is only useful when combined with **/F**.

/T (Time and date) Display the time and date for each directory. If you combine **/T** and **/F**, the time and date for each file will also be displayed. For LFN, HPFS, and NTFS files, the time and date of the last write will be shown by default. You can select a specific time and date stamp by using the following variations of **/T**:

/T:a last access date and time (access time is not saved on LFN volumes).

/T:c	creation date and time.
/T:w	last write date and time (default).

TRUENAME

Purpose: Find the full, true path and file name for a file.

Format: **TRUENAME *file***

file: The file whose name TRUENAME will report.

See also: @TRUENAME.

Usage

Default directories, as well as the JOIN and SUBST external commands, can obscure the true name of a file. TRUENAME "sees through" these obstacles and reports the fully qualified name of a file.

The following example uses TRUENAME to get the true pathname for a file:

```
[c:\] subst d: c:\util\test
[c:\] truename d:\test.exe
c:\util\test\test.exe
```

On LFN drives TRUENAME returns the short name for the file, for example:

```
[c:\] truename "Program Files"
C:\PROGRA~1
```

TRUENAME can handle simple drive substitutions such as those created by JOIN, SUBST, or most network drive mappings. However, it may not be able to correctly determine the true name if you use "nested" JOIN or SUBST commands, or a network which does not report true names properly.

TYPE

Purpose: Display the contents of the specified file(s).

Format: **TYPE [/A:[[-]rhsda] /L /P]** ***file*****...**

file: The file or list of files that you want to display.

/A: (Attribute select) **/P**(ause)
/L(ine numbers)

See also: LIST.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

The TYPE command displays a file. It is normally only useful for displaying ASCII text files. Executable files (*.COM* and *.EXE*) and many data files may be unreadable when displayed with TYPE because they include non-alphanumeric characters.

To display the files *MEMO1* and *MEMO2*:

```
[c:\] type /p memo1 memo2
```

You can press **Ctrl-S** to pause TYPE's display and then any key to continue.

To display text from the clipboard use **CLIP:** as the file name. CLIP: will not return any data if the clipboard does not contain text. See Redirection for additional information on CLIP:.

You will probably find LIST to be more useful for displaying files. However, the TYPE /L command used with redirection is useful if you want to add line numbers to a file, for example:

```
[c:\] type /l myfile > myfile.num
```

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only
H	Hidden
S	System
D	Subdirectory
A	Archive

If no attributes are listed at all (*e.g.*, **TYPE /A: ...**), TYPE will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/L (Line numbers) Display a line number preceding each line of text.

/P (Pause) Prompt after displaying each page. Your options at the prompt are explained in detail under Page and File Prompts.

UNALIAS

Purpose: Remove aliases from the alias list.

Format: **UNALIAS [/Q /R *file*...] [*alias*...]**

or

UNALIAS *

alias: One or more aliases to remove from memory.
file: One or more files to read for alias definitions.

/Q(uiet) **/R**(ead file)

See also: ALIAS and ESET.

Usage

4NT maintains a list of the aliases that you have defined. The UNALIAS command will remove aliases from that list. You can remove one or more aliases by name, or you can delete the entire alias list by using the command **UNALIAS ***.

For example, to remove the alias DDIR:

```
[c:\] unalias ddir
```

To remove all the aliases:

```
[c:\] unalias *
```

If you keep aliases in a file that can be loaded with the ALIAS /R command, you can remove the aliases by using the UNALIAS /R command with the same file name:

```
[c:\] unalias /r alias.lst
```

This is much faster than removing each alias individually in a batch file, and can be more selective than using UNALIAS *.

Options

/Q (Quiet) Prevents UNALIAS from displaying an error message if one or more of the aliases does not exist. This option is most useful in batch files, for removing a group of aliases when some of the aliases may not have been defined.

/R (Read) Read the list of aliases to remove from a file. The file format should be the same format as that used by the ALIAS /R command. You can use multiple files with one UNALIAS /R command by placing the names on the command line, separated by spaces:

```
[c:\] unalias /r alias1.lst alias2.lst
```

UNSET

Purpose: Remove variables from the environment or disable file associations inherited from Windows NT.

Format: **UNSET [/Q /R file...] *name*...**

or

UNSET *

name: One or more variables to remove from the environment or file types to disable.
file: One or more files containing variable definitions.

/Q(uiet) **/R**(ead from file)

See also: ESET and SET.

Usage

UNSET removes one or more variables from the environment, or disables file associations "inherited" from Windows NT.

For example, to remove the variable CMDLINE:

```
[c:\] unset cmdline
```

If you use the command **UNSET ***, all of the environment variables will be deleted:

```
[c:\] unset *
```

UNSET can be used in a batch file, in conjunction with the SETLOCAL and ENDLOCAL commands, to clear the environment of variables that may cause problems for applications run from that batch file.

For more information on environment variables, see the SET command and the general discussion of the environment.

You can also use UNSET to disable direct file associations that 4NT has inherited from Windows NT (see Windows File Associations and Using Windows File Associations for more details on inherited file associations). If the first character of the variable name is a period [**.**], UNSET will look first for a matching environment variable to remove. If it doesn't find one, it will next look in the list of direct file associations it has loaded from Windows NT. UNSET will not modify Windows NT's file associations, just 4NT's copy of the associations.

Use caution when removing environment variables, and especially when using UNSET *. Many programs will not work properly without certain environment variables; for example, 4NT uses PATH and DPATH.

Options

/Q (Quiet) Prevents UNSET from displaying an error message if one or more of the variables or associations does not exist. This option is most useful in batch files, for removing a group of variables when some of the variables may not have been defined.

/R (Read) Read environment variables to UNSET from a file. This much faster than using multiple UNSET commands in a batch file, and can be more selective than UNSET *. The file format should be the same format as that used by the SET /R command.

VER

Purpose: Display the current command processor and operating system versions.

Format: **VER [/R]**

/R(evision level)

Usage

Version numbers consist of a one-digit major version number, a separator, and a one- or two-digit minor version number. VER uses the default decimal separator defined by the current country information. The VER command displays both version numbers:

```
[c:\] ver
4NT 3.00A   Windows NT Version is 4.0
```

Options

/R (Revision level) Display the 4NT and Windows NT internal revision levels, plus your 4NT serial number and registered name.

VERIFY

Purpose: Enable or disable disk write verification or display the verification state.

Format: **VERIFY [ON | OFF]**

Usage

Disk write verification cannot actually be enabled or disabled under Windows NT. 4NT supports VERIFY as a "do-nothing" command, for compatibility with *CMD.EXE*. This avoids "unknown command" errors in batch files which use the VERIFY command.

VOL

Purpose: Display disk volume label(s).

Format: **VOL [*d*:] ...**

d: The drive or drives to search for labels.

Usage

Each disk may have a volume label, created when the disk is formatted or with the external LABEL command. Also, every floppy disk formatted with DOS version 4.0 or above, Windows NT, or Windows NT has a volume serial number.

The VOL command will display the volume label and, if available, the volume serial number of a disk volume. If the disk doesn't have a volume label, VOL will report that it is "unlabeled." If you don't specify a drive, VOL displays information about the current drive:

```
[c:\] vol
Volume in drive C: is MYHARDDISK
```

If available, the volume serial number will appear after the drive label or name.

To display the disk labels for drives A and B:

```
[c:\] vol a: b:
Volume in drive A: is unlabeled
Volume in drive B: is BACKUP_2
```

VSCRPUT

Purpose: Display text vertically in the specified color.

Format: **VSCRPUT** ***row col*** **[BRIght]** ***fg*** **ON [BRIght]** ***bg text***

row: Starting row number.
col: Starting column number.
fg: Foreground text color.
bg: Background text color.
text: The text to display.

See also: SCRPUT.

Usage

VSCRPUT writes text vertically on the screen rather than horizontally. Like the SCRPUT command, it uses the colors you specify to write the text. VSCRPUT can be used for simple graphs and charts generated by batch files.

The *row* and *column* are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. VSCRPUT checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move down the specified number of rows or to the right the specified number of columns before displaying text, or with a minus sign [**-**] to move up or to the left.

If you specify 999 for the *row*, VSCRPUT will center the text vertically on the display. If you specify 999 for the *column*, VSCRPUT will center the text horizontally.

VSCRPUT normally does not move the cursor when it displays the *text*.

The following batch file fragment displays an X and Y axis and labels them:

```
cls bright white on blue
drawhline 20 10 40 1 bright white on blue
drawvline 2 10 19 1 bright white on blue
scrput 21 20 bright red on blue X axis
vscrput 8 9 bright red on blue Y axis
```

WINDOW

Purpose: Minimize or maximize the current window, restore the default window size, set the window size or position, or change the window title.

Format: **WINDOW [MIN | MAX | RESTORE | /POS=x, y, width, height | /SIZE=rows, columns | "*title* "]**

title: A new title for the window.

/POS(ition) **/SIZE** (of screen buffer)

Usage

WINDOW is used to control the appearance and title of the current window. WINDOW MIN reduces the window to an icon, WINDOW MAX enlarges it to its maximum size, and WINDOW RESTORE returns the window to its default size and location on the desktop.

You can use the **/POS** option to set the location and size of the window on the desktop. The x and y values of the **/POS** option select the window's origin (from the bottom left of the screen) while the width and height values determine its size.

If you specify a new title, the title text must be enclosed in double quotes. The quotes will not appear as part of the actual title.

You can only specify one WINDOW option at a time. The different options cannot be combined in a single WINDOW command. To perform multiple operations, use multiple WINDOW commands.

Options

/POS Set the window screen position and size. The syntax is **/POS=x, y, width, height**, where the values are specified in pixels or pels. **x** and **y** refer to the position of the bottom left corner of the window relative to the bottom left corner of the screen.

/SIZE Specify the screen buffer size. The full syntax is **/SIZE=rows, columns**, where **rows** is the number of text rows and **columns** is the number of text columns. Due to the design of Windows NT console sessions, you cannot use **/SIZE** to reduce the size of the screen buffer; it can only be increased.

Y

Purpose: Copy standard input to standard output, and then copy the specified file(s) to standard output.

Format: **Y *file* ...**

file: The file or list of files to send to standard output.

See also: TEE.

Usage

The Y command copies input from standard input (usually the keyboard) to standard output (usually the screen). Once the input ends, the named files are appended to standard output.

For example, to get text from standard input, append the files *MEMO1* and *MEMO2* to it, and send the output to *MEMOS*:

```
[c:\] y memo1 memo2 > memos
```

The Y command is most useful if you want to add redirected data to the beginning of a file instead of appending it to the end. For example, this command copies the output of DIR, followed by the contents of the file DIREND, to the file DIRALL:

```
[c:\] dir | y dirend > dirall
```

If you are typing at the keyboard to produce input text for Y, you must enter a **Ctrl-Z** to terminate the input.

When using Y with a pipe you must take into account that the programs on the two ends of the pipe run simultaneously, not sequentially.

See Piping for more information on pipes.

Starting 4NT

You will typically start 4NT from an object on your Windows NT desktop. You can create as many 4NT objects as you wish on the desktop. Different objects can be used to start 4NT in different modes, with different startup commands or options, or to run different batch files or other commands. You can use these objects to run commonly-used commands and batch files directly from the Windows NT desktop.

Each object represents a different 4NT window. You can set any necessary command line parameters for 4NT such as a command to be executed, any desired switches, and the name and path for *4NT.INI*. More information on command line switches and options for 4NT is included later in this section.

For general information on creating and configuring desktop objects, see your Windows NT documentation.

When you configure a 4NT object, place the full path and name for the *4NT.EXE* file in the Command Line field, and put any startup options that you want passed to 4NT (*e.g.*, **@inifile**) after the *4NT.EXE* file name. For example:

```
Command Line:        D:\4NT300\4NT.EXE @D:\4NT.INI
Working directory:   C:\
```

To run a startup batch file for a particular 4NT window, include its name (with a path, if the batch file is not in the startup directory) as the last item in the Command Line field. That batch file will be executed after any *4START* file but before the first prompt is displayed. You can use the batch file to set environment variables and execute any other 4NT commands. You can also execute any internal 4NT command, external command, or alias by placing its name in the Command Line field. When you set up a batch file or other command to run in this way you are using the **command** option (see below). For example:

```
Command Line:        D:\4NT300\4NT.EXE STARTNT.CMD
Working directory:   C:\
```

To execute an internal or external command, an alias, or a batch file and then exit (return to the desktop) when it is done, place **/C command** (rather than just **command**) as the last item in the Command Line field. For example:

```
Command Line:        D:\4NT300\4NT.EXE /C COMFILES.BTM
Working directory:   C:\
```

The 4NT command line does not need to contain any information. When invoked with an empty command line, 4NT will configure itself from the *4NT.INI* file, run *4START*, and then display a prompt and wait for you to type a command. However, you may add information to the 4NT command line that will affect the way it operates.

Command line options for primary shells are set in the Command Line field of the 4NT object. Command line options for secondary shells can be set on the secondary shell command line.

4NT recognizes several optional fields on the command line. All of the options go on one line. If you use more than one of these fields, their order is important. The syntax for the command line is:

[d:\path] [@d:\path\inifile] [//iniline]... [/L] [/LA] [/LD] [/LH] [/Q] [/S] [/T:bf] [X] [/C | /K] [command]

The options are:

d:\path: This option sets the drive and directory where the program is stored, called the

COMSPEC path. 4NT uses this path to find its files and to set the COMSPEC environment variable. 4NT normally knows what drive and directory it is started from, so this option is not usually necessary.

@d:\path\inifile: This option sets the path and name of the *4NT.INI* file. You do not need this option if you aren't using a *4NT.INI* file, or if the file is named *4NT.INI* and it is either in the same directory as *4NT.EXE* or in the root directory of the boot drive. This option is most useful if you want to start the program with a specific and unique *.INI* file.

//iniline: This option tells 4NT to treat the text appearing between the *//* and the next space or tab as an *.INI* directive. The directive should be in the same format as a line in *4NT.INI*, but it may not contain spaces, tabs, or comments. Directives on the command line override any corresponding directive in the *.INI* file. This is a convenient way to place one or two simple directives on the startup line without having to modify or create a new *.INI* file.

/L, **/LA**, **/LD**, and **/LH**: These options force 4NT to to use a local alias, directory history, and / or command history list. They can be used to override any LocalAliases=No, LocalDirHistory=No, or LocalHistory=No settings in *4NT.INI*. This allows you to use global lists as the default, but start a specific 4NT session with local aliases or histories.

See Command History for details on local and global history, Directory History Window for details on local and global directory history, and ALIAS for details on local and global aliases. **/LA** forces local aliases, **/LD** forces local directory history, **/LH** forces local command history, and **/L** forces all three: local aliases, command history, and directory history.

/Q: This option has no effect. It is included only for compatibility with *CMD.EXE*.

/S: This option tells 4NT that you do not want it to set up a Ctrl-C / Ctrl-Break handler. It is included for compatibility with *CMD.EXE*, but it may cause the system to operate incorrectly if you use this option without other software to handle Ctrl-C and Ctrl-Break. This option should be avoided by most users.

/T:bf: This option sets the foreground and background colors in the 4NT window. Both **b** and **f** are hexadecimal digits; **b** specifies the background color and **f** specifies the foreground color. This option is included only for compatibility with *CMD.EXE*; in most cases you should set default colors with the StdColors directive in *4NT.INI*, or the corresponding Output Colors option on the Display page of the OPTION dialogs. If you use both, the **/T** switch overrides any StdColors setting.

/X: This option forces 4NT to alter the operation of the MD and MKDIR command to automatically create all necessary intermediate directories when it creates a new subdirectory. Its effect is the same as adding a /S option to all MD and MKDIR commands. (This option is included for compatibility with *CMD.EXE*. In *CMD.EXE* it enables other options as well, but in 4NT the only option not enabled by default is the implicit MD /S.)

[/C | /K] command: This option tells 4NT to run a specific command after starting. The command will be run after 4START has been executed and before any command prompt is displayed. It can be any valid alias, internal or external command, or batch file. All other startup options must be placed before the command, because 4NT will treat characters after the command as part of the command and not as additional startup options.

When the command is preceded by a **/C**, 4NT will execute the command and then exit and return to the parent program or the Windows NT desktop without displaying a prompt. This is sometimes called a "transient" command interpreter session.

The **/K** switch has no effect; using it is the same as placing the command (without a **/C** or **/K**) at the end of the startup command line.   It is included only for compatibility with *CMD.EXE*.

What's New?

There are dozens of new features in this latest release of 4NT!   Check through this section for a comprehensive list of what's changed since our previous release, version 2.52.

This topic does not explain how to use each new feature.   Instead, where appropriate we have provided links below to the detailed help topics containing additional usage information or other documentation.

Some of the descriptions here may be more detailed than you need; if you aren't using a feature, feel free to skip to the next item.   If you are new to 4NT with version 3.0, you can skip this topic entirely.

This topic is divided into the following subtopics:

» General Features and Enhancements

» Command Line Editing

» Command Changes

» Variables and Variable Functions

» Startup and Configuration

» Technical and Compatibility Enhancements

» Bugs Fixed

The major new features in this release include:

» **Extended Directory Searches** allow you to change to a directory anywhere on your system by entering only part of its name.   They must be explicitly enabled before you can use them. See Directory Navigation for complete details.

» You can **directly execute URLs, and files with Windows file associations**, from the 4NT prompt.

» **New commands** include:

ECHOERR and **ECHOSERR**:   Display output on the "standard error" device (rather than the usual "standard output" device).

OPTION: Offers complete configuration adjustment, either through interactive dialogs or on the command line.

SWITCH: Provides for "case" statements in batch files.

TOUCH: Adjusts file dates and times.

TREE: Displays the directory tree, with or without file names, in a variety of formats.

» New **file exclusion ranges** provide a convenient way to exclude files from any internal command -- faster and more flexible than using EXCEPT.

» The new **batch file debugger** can execute each line step by step, process or trace into additional batch files, and display variables, aliases, and expanded commands at each step.

There over 100 additional new features beyond those mentioned here.   See the individual subtopics listed above for details.

What's New: General Features and Enhancements

» Added a complete batch file debugger.   The debugger displays the batch file in a window and allows you to execute each line step by step, process or trace into additional batch files and subroutines, and display variables and aliases at each step.   See Batch File Debugging for complete details.

» You can now use Windows NT file associations to execute files directly from the prompt.   In other words, you can directly execute something like "myfile.doc" without having to define an executable extension, as long as the file has an associated application.   You can also directly execute *.LNK* files and URLs at the prompt.

» Added a new type of range called a "file exclusion range".   The syntax is "/[!filename ...]" which excludes filenames that match those inside the brackets.   For example, to display everything but *.TXT* and *.BAK* files:

```
          dir /[!*.txt *.bak] *.*
```

Exclusion ranges are faster, more flexible, and more reliable than the similar EXCEPT command when excluding files from processing by internal commands.   However they do not work with external commands.

» Popup windows (for filename completion, command history recall, etc.) now allow you to search for a line within the window contents by typing the first few characters of the line.   The search string is displayed in the lower right corner of the window.

» You can now redirect to and from the clipboard by using the pseudo-device name CLIP:.   For example, to redirect DIR to the clipboard:

```
   dir *.doc > clip:
```

» The online help now includes a standard Windows NT Table of Contents.   Also the help has been reorganized to make it easier to navigate through the main topics, and includes additional reference information, reference tables, and a glossary.

» The default maximum file description length is now 511 bytes in all products.

» Two new characters can now follow the escape character: An escape followed by a 'q' will substitute a double quote; an escape followed by a 'k' will substitute a back quote.

» The decimal and thousands characters used in @EVAL and in displayed version numbers and other similar locations are now controllable with the DecimalChar and ThousandsChar directives in the *.INI* file, the corresponding options in the configuration or OPTION dialogs, and the SETDOS /G command.   These characters are saed by SETLOCAL and restored by ENDLOCAL.   This is intended as an aid to those writing batch files which perform arithmetic operations and which may be used in countries with differing separator characters.

» The directory stack size used by PUSHD and POPD has been increased from 255 to 511 bytes to leave adequate room for long directory names.

» *.BTM* files can now be longer than 64K bytes, though compressed *.BTM*s still have to be less than 64K.

[This subtopic covers some of the new features in this version of 4NT.   For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Command Line Editing

» Extended directory searches can be used directly from the command line for quick directory navigation; see Automatic Directory Changes or Directory   Navigation for details.

» Made several enhancements to Filename completion. including:

  º The Ctrl-A key, which toggles between long and short filenames for filename completion, can now be hit at any point during command line entry -- not just during filename completion.   For example, if you hit Ctrl-A at the beginning of the command line, all filenames subsequently returned for that line will be short names (until you hit Ctrl-A again).

  º Filename completion can now be customized for individual commands via the new FileCompletion *.INI* directive (or environment variable).   For example, you can configure 4NT to complete only the names of *.TXT* files when the command line starts with the name of your text editor, or to display only directory names when you are entering a CD command.

  º The F7 filename completion popup window now sorts the filename list alphabetically.

» You can now expand aliases immediately while still on the command line with the Ctrl-F key.

» Command line history recall will now stop at the beginning and end of the history list rather than wrapping around, if you set HistWrap to No in the *.INI* file or through the configuration dialogs.

[This subtopic covers some of the new features in this version of 4NT.   For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Command Changes

» ASSOC: This new command is included for compatibility with *CMD.EXE*. It assists you in managing the relationships between file extensions and file types stored in the Windows registry.

» ATTRIB: Added the /E switch to disable display of non-fatal errors. Also, ATTRIB now allows underscores in the attribute string, so that you can get a result from the @ATTRIB variable function and feed it directly to the ATTRIB command.

» CD and CDD: Now support extended directory searches, which allow you to change to a directory anywhere on your system by entering only part of its name. The CDD /S switch builds the extended directory search database. Extended directory searches mmust be explicitly enabled before you can use them. See Directory Navigation for complete details.

» CD: Added support for Windows NT 4.0's undocumented CD /D switch to change drive and directory (like the 4NT CDD command).

» CDD: Added the /A switch to display the current directory for all existing and ready drives from C: to Z:.

» CLS: Changed the basic CLS command to clear the screen, not the entire screen buffer; added a /C switch to clear the entire buffer.

» COLOR: Added support for *CMD.EXE*'s syntax ("COLOR xy", where x is the background color in hex, and y is the foreground color in hex).

» COPY: Added several switches:

/E	Disable display of non-fatal errors.
/K	Preserve read-only attributes during a COPY.
/X	Clear the archive bit from the source file after a successful copy.
/Z	Overwrite read-only target files.

» DATE: Added support for *CMD.EXE*'s /T switch, which displays the date without prompting for input.

» DEL: Added two switches:

/E	Disable display of non-fatal errors.
/W	Clear the file to 0's before deleting it.

» DIR: Added or modified several of the DIR switches:

/2	Now forces use of the short name on LFN drives.
/4	Now forces use of the short name on LFN drives, and displays files between 1 and 9.9 Mb in tenths (i.e., "2.4M").
/C	(New) Shows compression percentage on compressed NTFS drives.
/G	(New) Displays the allocated size instead of the file size.
/W	Now forces use of the short name on LFN drives.

/X Now shows the full short pathname when used with /F.

» DIRHISTORY: This new command has the same syntax as HISTORY, but it modifies the directory history.

» DO: Added two new DO loop types:

  º "DO x IN filename" retrieves each matching filename from a wildcard spec and inserts the value into the variable.

  º "DO x IN @filename" retrieves each line in the file and inserts it into the variable.

» ECHOERR and ECHOSERR: These new commands are like ECHO and ECHOS, but output goes to the standard error device instead of standard output.

» ENDLOCAL: To aid in making batch files portable, SETLOCAL and ENDLOCAL now save and restore the command separator, escape character, parameter character, and decimal and thousands separators.

» FFIND: Added two new switches:

  /I Do a literal match even if the text search string contains wildcard characters.

  /R Start searching for text from the end backwards.

  Also, the /X switch will now display the offset in both hex and decimal.

» FOR: Added several new switches for compatibility with Windows NT 4.0's *CMD.EXE*; see the command reference information for complete details.

» FTYPE: This new command is included for compatibility with *CMD.EXE*. It assists you in managing the relationships between file extensions and file types stored in the Windows registry.

» GOTO: Added support for Windows NT 4.0's "GOTO :EOF" -- If there is no ":EOF" label, GOTO ends the current batch file (equivalent to a QUIT).

» IF / IFF: These commands have several changes, including:

  º Support for nested conditional tests, with parentheses, *e.g.*:

```
if (%a == 1 .or. %b == 2) .and. %c == 3 echo something
```

  See the command reference information for complete syntax rules.

  º A new "IF DEFINED varname" test, which succeeds if the specified variable exists in the environment. This is included for compatibility with Windows NT 4.0's *CMD.EXE*, and is the same as a test like:

```
if "%varname" ne "" ...
```

  º The comparison tests now accept a leading decimal separator as a numeric character, provided the remainder of the string is numeric and does not contain additional decimal characters.

  º New options for compatibility with Windows NT 4.0's *CMD.EXE*:

    - Conditional tests "eql", "neq", "lss", "leq", "gtr", and "geq" (equivalent to "eq", "ne", etc.)

- The /I(gnore) case switch; this switch does nothing in 4NT, which is already case-insensitive

º A new ISWINDOW test to check for existence of a window.

» LIST: Add a range of enhancements, including:

º Added three new switches:

/I	Ignore case in a /T search.
/R	The search initiated by /T goes backwards from the end of the file.
/T	Search for text when LIST starts.

º Ctrl-PgUp and Ctrl-PgDn will go to the previous and next file in the current group, respectively.

º Ctrl-F searches backwards for a text string; Ctrl-N repeats the last search, searching backwards.

º Matching strings on the first page are now highlighted after a search.

º When piping output to LIST in most cases you no longer need the /S switch; for example, to view DIR's output in LIST you can now use:

```
dir | list
```

» MD: Added the /N switch to create a directory without updating the extended directory search database (useful for temporary directories).

» MOVE: Added the /E switch to disable display of non-fatal errors.

» OPTION: This new command can be used for two purposes. When invoked without parameters, it loads configuration dialogs which adjust most commonly-used settings in the *.INI* file. The dialogs provide a convenient method of adjusting configuration without manually editing the *.INI* file. OPTION can also be used to change specific settings on an individual basis with the OPTION Name=value ... syntax; see the command for complete details.

» PROMPT: Added the $+ metacharacter, which displays one + for each PUSHD level.

» RD: The /S switch included for compatibility with *CMD.EXE* no longer deletes files and directories without prompting. It is just as destructive as before, but -- like *CMD.EXE* in Windows NT 4.0 -- it prompts first.

» REN / RENAME: Added the /E switch to disable display of non-fatal errors.

» RETURN: Now accepts an optional argument for the errorlevel to return. The errorlevel can be tested with %? or IF ERRORLEVEL.

» SCREEN, SCRPUT, and VSCRPUT: If you specify 999 for the row, the text will be centered vertically; if you specify 999 for the column, the text will be centered horizontally.

» SELECT: You can now type characters from the start of a filename and the selection bar will jump to the first matching name. Due to this change, the key to popup LIST on the currently selected file has been changed from L to ^L. Also, added the /T:acw switch to select the date and time to use for display and sorting on LFN, HPFS,and NTFS drives.

» Changed the /Z switch to display short filenames (as DIR does) rather than just truncating the name.

» SET: Added the /A switch for compatibility with *CMD.EXE*. /A evaluates the argument to the right of the equal sign as an arithmetic expression, using the same rules as @EVAL, and displays the result as well as storing it in an environment variable..

» SETLOCAL: To aid in making batch files portable, SETLOCAL now saves the command separator, escape character, parameter character, and decimal and thousands separators; ENDLOCAL restores them.

» SHIFT: The new "/n" argument will start the shift at the specified argument -- i.e., "shift /2" moves %3 to %2, %4 to %3, etc.

» START: Added /LD for a local directory history list, and the /SHARED option.

» START: If the program name to execute is actually a directory name instead of an executable, START runs Windows Explorer in the specified directory.

» SWITCH: This new command provides a C-like switch construct for batch files. SWITCH scans each CASE statement looking for a matching value; if it finds one it executes the block of code inside that CASE statement, and then jumps to the end of the switch block (ENDSWITCH). If no CASE statement matches, SWITCH will execute the code in the (optional) DEFAULT block.

» TIME: Added support for *CMD.EXE*'s /T switch, which displays the time without prompting for input.

» TOUCH: This new command changes the date and/or time for a file or files. You can set a specified date and time or use the current system clock, and you can optionally change the last access / creation date and time fields on LFN, HPFS, and NTFS drives.

» TREE: This new command displays a graphical directory tree using either line-drawing or ASCII characters. It can also optionally display file names, dates, times, and sizes.

» UNALIAS: Added the /R switch to read a file of aliases to remove.

» UNSET: Added the /R switch to read a file of variables to remove.

[This subtopic covers some of the new features in this version of 4NT. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Variables and Functions

Added or updated the following internal variables (all variables listed are new unless otherwise noted):

» _CPU: Now returns "686" for Pentium Pro.

» _DOWI: Returns the current day of week as an integer (Sun = 1, Mon = 2, etc.).

» ERRORLEVEL: Returns the exit code from the last external program (same value as the "?" variable).

Added or updated the following variable functions (all functions listed are new unless otherwise noted):

» @CLIP[n]:   Returns line **n** from the clipboard (base 0).

» @CONVERT[input,output,value]:   Converts a number from one base to another.

» @DAY[date]:   Returns the day for the specified date.

» @DOW[date]:   Returns the day of week for the specified date, as a string (Sun, Mon, etc.)

» @DOWI[date]:   Returns the day of week for the specified date, as an integer (Sun = 1, Mon = 2, etc.).

» @DOY[date]:   Returns the day of year for the specified date (1-366).

» @EVAL[expression]:   Now supports user-definable decimal and thousands characters; see DecimalChar and ThousandsChar, or SETDOS /G for details.

» @EXEC[command]:   This function has been modified; if you preface the command with an '@', @EXEC will return an empty string rather than the result code of the command.

» @EXECSTR[command]:   Returns the first line written to STDOUT by the specified command.   (This is intended to provide functionality similar to UNIX back-quoting.)

» @EXPAND[filename[,attributes]]:   Expands a wildcard filename and returns all of the matching filenames / directory names on a single line.

» @FILEDATE[filename[,acw]] / @FILETIME[filename[,acw]]:   Added the optional second argument to determine which date / time field to return on LFN, HPFS, and NTFS drives.

» @FILESIZE[filename[,bkm[,a]]:   Added the optional third argument **a**(llocated); if specified, the function returns the size actually used on disk, not the amount of data in the file.

» @INIREAD[filename,section,entry]:   Reads a setting from a *.INI* file.

» @INIWRITE[filename,section,entry,string]:   Writes a setting to a *.INI* file.

» @INSERT[n,string1,string2]:   Inserts **string1** into **string2** starting at offset **n**.

» @LEFT[n,string]:   Returns the leftmost **n** characters of **string**.

» @LFN[filename]:   Returns the long version of a short ("8.3") filename.   The return value includes the full path.

- » @MONTH[date]: Return the month for the specified date.
- » @NUMERIC[string]: Now considers a leading decimal separator as a numeric character, provided the remainder of the string is numeric and does not contain additional decimal characters.
- » @REPLACE[string1,string2,text]: Replaces all occurrences of **string1** in text with **string2**.
- » @RIGHT[n,string]: Returns the rightmost **n** characters of **string**.
- » @SEARCH[filename[,path]]: Now accepts an optional second argument for the path to search.
- » @SELECT[filename,top,left,bottom,right,title[,1]]: Has a new optional argument following the title. If it's set to 1, @SELECT will sort the list alphabetically.
- » @SFN[filename]: Returns the short ("8.3") version of a long filename. The return value includes the full path.
- » @STRIP[chars,string]: Return **string** with the characters in **chars** removed.
- » @WILD[string1,string2]: Does a wildcard comparison on the two strings and returns 1 if they match; 0 if they don't.
- » @YEAR[date]: Return the year for the specified date.

[This subtopic covers some of the new features in this version of 4NT. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Startup and Configuration

» In previous versions the "global" portion of the .INI file (the part prior to any [Primary] or [Secondary] section) did not have a section name.   This has been changed; a section name matching the product name is now required, for example:

```
[4NT]
EditMode = Insert
.....
```

[Primary] and [Secondary] section names are still supported as well.

Added or modified the following *.INI* directives (all are new unless otherwise noted):

» CDDWinLeft, CDDWinTop, CDDWinWidth, CDDWinHeight, CDDWinColors: These directives set the position, size, and color of the popup window used for extended directory searches.

» DuplicateBugs = Yes | NO: Tells the parser to duplicate certain CMD.EXE errors which may be important in solving rare compatibility problems.   The only bug currently replicated by this command is the IF command.

» FileCompletion = cmd1:ext1 ext2;cmd2 ...:   Sets up command-specific filename completion.

» FuzzyCD =   0 | 1 | 2 | 3:   Enables or disables extended directory searches, and controls their behavior.

» HistMove = Yes | NO: If set to Yes, a recalled line from the command history is moved to the end of the history list, and removed from its original location.

» Include = filename: Includes the contents of the named file as if they had appeared at the location of the Include= directive in the current *.INI* file.

» ListboxBarColors = Color: Sets the color for the highlight bar in the popup listboxes (command history, filename completion, @SELECT, etc.).

» LoadAssociations = YES | No: Determines whether the command processor will load Windows file associations at startup.

» TabStops = nnnn (8):   This new *.INI* directive specifies the tab expansion size when displaying a file in LIST.

» TreePath = Path: Specifies the location of *JPSTREE.IDX* (the extended directory search database; defaults to C:\).

[This subtopic covers some of the new features in this version of 4NT.   For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Technical and Compatibility Enhancements

» Added support for Enterprise REXX (in addition to the existing Quercus REXX support).

» Added support for the *CMD.EXE* /T switch to set colors on the startup command line.

» Changed the executable file search order.   4NT will first look for *.COM*, *.EXE*, *.BAT*, *.CMD*, and user-defined executable extensions in the current directory and the PATH.   If not found, it will search the path again for system-defined extensions (file associations).   This is for compatibility with *CMD.EXE*.

» Improved support for drive and file sizes over 4GB.

» The title bar is now updated for internal commands as well as externals, for compatibility with *CMD.EXE* in Windows NT 4.0.

» TYPE NUL now "works" (i.e. it generates no output), for compatibility with batch files which use TYPE NUL > file to generate a 0-byte file.

» Worked around a Windows NT bug that caused 4NT to discard the first token on the passed command line under certain circumstances when it was started by another application.

» Added debugging options which allow you to view the command "tail" passed to Take Command, and to "tag" error messages with the product name.   See the Debug directive in *4NT.INI* for additional details.

[This subtopic covers some of the new features in this version of 4NT.   For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

What's New: Bugs Fixed

- » Piping the output of a batch file which also contains a pipe will no longer cause problems.
- » Fixed a problem with piping which occurred only when the process on the right hand side of the pipe completed very quickly.
- » DESCRIBE: Fixed a problem with quoted long filenames with paths.
- » RENAME: Now works properly when renaming quoted long filenames with embedded wildcards.
- » START: Now supports a quoted long directory name after the /D switch. Also, changed the order of parameters for the /POS option to match the documentation (left, top, width, height).
- » ExecWait now works properly.
- » Fixed a problem with executable extensions when the application name or path included embedded whitespace.
- » Fixed a problem which could cause trouble in a transient session if the COMSPEC directory name was present on the command line after a /C (e.g. 4NT /C echo COMSPEC is %COMSPEC).
- » Quoted long filenames can now be used in the .INI file.
- » @FILESEEKL now always returns to the start of the file before seeking.
- » Added some protection to description handling to avoid destroying descriptions for files with long names when updating the description file in a non-LFN environment.

[This subtopic covers some of the new features in this version of 4NT. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

Command-Line Help

This online help system for 4NT covers all 4NT features and internal commands. It includes reference information to assist you in using 4NT and developing batch files, and it includes most — but not all — of the details which are included in the printed 4NT manuals.

You can start the online help system at the command line by entering **HELP** or **HELP** plus a topic, or by pressing the **F1** key at any time.

If you have already typed part or all of a command on the line, the help system will provide "context-sensitive" help by using the first word on the line as a help topic. If it's a valid topic, you will see help for that topic automatically; if not, you will see a table of contents and you can then pick the topic you want. For example, if you press **F1** after entering each of the command lines shown below you will get the display indicated:

```
[c:\]                    Topic list / table of contents
[c:\] copy *.* a:        Help on COPY
[c:\] c:\util\map        Topic list / table of contents
```

For help you can also type the name of any internal command at the prompt, followed by a slash and a question mark [**/?**] like this:

```
[c:\] copy /?
```

The **/?** option may not work correctly if you have redefined how the command operates with an alias. In this case you may need to add an asterisk to the beginning of the command to disable alias processing:

```
[c:\] alias copy copy /r
[c:\] *copy /?
```

/? will only access the help system when you use it with an internal command. If you use it with an external command name, the external command will be executed and will interpret the **/?** parameter according to its own rules. Some external commands do display help when run with a **/?** parameter, but this a characteristic of these commands and does not depend on the command processor. Many other external commands do not have this feature.

4NT uses the Windows NT help system to display this help text. Once you've started the help system with HELP or **F1**, you can use standard Windows NT keystrokes to navigate. For more information, click on the Help menu at the top of this window.

Error Messages

A B C D E F G I K L M N O R S T U V W

This section lists error messages generated by 4NT, and includes a recommended course of action for most errors. If you are unable to resolve the problem, look through your Introduction and Installation Guide for any additional troubleshooting recommendations, then contact JP Software for technical support.

Error messages relating to files are generally reports of errors returned by Windows NT. You may find some of these messages (for example, "Access denied") vague enough that they are not always helpful. 4NT includes the file name in file error messages, but is often unable to determine a more accurate explanation of these errors. The message shown is the best information available based on the error codes returned by Windows NT.

For some errors you are instructed to "restart the session or reboot the system." This means that you should attempt to correct the error by closing and restarting the current session under Windows NT.

The following list includes all error messages, in alphabetical order:

Access denied: You tried to write to or erase a read-only file, rename a file or directory to an existing name, create a directory that already exists, remove a read-only directory or a directory with files or subdirectories still in it, or access a file in use by another program in a multitasking system.

Alias loop: An alias refers back to itself either directly or indirectly (*i.e.*, a = b = a), or aliases are nested more than 16 levels deep. Correct your alias list.

Already excluded files: You used more than one exclude range in a command. Combine the exclusions into a single range.

Bad disk unit: Generally caused by a disk drive hardware failure.

Batch file missing: 4NT can't find the batch (*.BTM* or *.CMD*) file it was running. It was either deleted, renamed, moved, or the disk was changed. Correct the problem and rerun the file.

Can't copy file to itself: You cannot COPY or MOVE a file to itself. 4NT attempts to perform full path and filename expansion before copying to help ensure that files aren't inadvertently destroyed.

Can't create: 4NT can't create the specified file. The disk may be full or write protected, or the file already exists and is read-only, or the root directory is full.

Can't delete: 4NT can't delete the specified file or directory. The disk is probably write protected.

Can't get directory: 4NT can't read the directory. The disk drive is probably not ready.

Can't make directory entry: 4NT can't create the filename in the directory. This is usually caused by a full root directory. Create a subdirectory and move some of the files to it.

Can't open: 4NT can't open the specified file. Either the file doesn't exist or the disk directory or File Allocation Table is damaged.

Can't remove current directory: You attempted to remove the current directory, which Windows NT does not allow. Change to the parent directory and try again.

CD-ROM door open or **CD-ROM not ready**: The CD-ROM drive door is open, the power is off, or the drive is disconnected.   Correct the problem and try again.

CD-ROM not High Sierra or ISO-9660:   The CD-ROM is not recognized as a data CD (it may be a music CD).   Put the correct CD in the drive and try again.

Clipboard is empty or not text format: You tried to retrieve some text from the Windows NT clipboard, but there is no text available.   Correct the contents of the clipboard and try again.

Clipboard is in use by another program: 4NT could not access the Windows NT clipboard because another program was using it.   Wait until the clipboard is available, or complete any pending action in the other program, then try again.

Command line too long: A single command exceeded 1023 characters, or the entire command line exceeded 2047 characters, during alias and variable expansion.   Reduce the complexity of the command or use a batch file.   Also check for an alias which refers back to itself either directly or indirectly.

Command only valid in batch file: You have tried to use a batch file command, like DO or GOSUB, from the command line or in an alias.   A few commands can only be used in batch files (see the individual commands for details).

Contents lost before copy: COPY was appending files, and found one of the source files is the same as the destination.   That source file is skipped, and appending continues with the next file.

Data error: Windows NT can't read or write properly to the device. On a floppy drive, this error is usually caused by a defective floppy disk, dirty disk drive heads, or a misalignment between the heads on your drive and the drive on which the disk was created. On a hard drive, this error may indicate a drive that is too hot or too cold, or a hardware problem.   Retry the operation; if it fails again, correct the hardware or diskette problem.

Directory stack empty: POPD or DIRS can't find any entries in the directory stack.

Disk is write protected: The disk cannot be written to.   Check the disk and remove the write-protect tab or close the write- protect window if necessary.

Drive not ready -- close door: The removable disk drive door is open.   Close the door and try again.

Duplicate redirection: You tried to redirect standard input, standard output, or stand error more than once in the same command.   Correct the command and try again.

Environment already saved: You have already saved the environment with a previous SETLOCAL command.   You cannot nest SETLOCAL / ENDLOCAL pairs.

Error in command-line directive: You used the **//iniline** option to place an *.INI* directive on the startup command line, but the directive is in error.   Usually a more specific error message follows, and can be looked up in this list.

Error on line [nnnn] of [filename]:   There is an error in your *4NT.INI* file.   The following message explains the error in more detail.   Correct the line in error and restart 4NT for your change to take effect.

Error reading: Windows NT experienced an I/O error when reading from a device.   This is usually caused by a bad disk, a device not ready, or a hardware error.

Error writing: Windows NT experienced an I/O error when writing to a device.   This is usually caused by a full disk, a bad disk, a device not ready, or a hardware error.

Exceeded batch nesting limit: You have attempted to nest batch files more than 10 levels deep.

File Allocation Table bad: Windows NT can't access the FAT on the specified disk. This can be caused by a bad disk, a hardware error, or an unusual software interaction.

File association not found: The ASSOC command could not find a file association for the specified extension in the Windows NT registry.

File exists: The requested output file already exists, and 4NT won't overwrite it.

File is empty: You attempted to use an empty file in @SELECT. Correct the file name or contents and try again.

File not found: 4NT couldn't find the specified file. Check the spelling and path name.

File type not found: The FTYPE command could not find the specified file type in the Windows NT registry.

General failure: This is usually a hardware problem, particularly a disk drive failure or a device not properly connected to a serial or parallel port. Try to correct the problem, or reboot and try again. Also see **Data error** above.

Include file not found: You used the Include directive in the 4NT.INI file, but the file you specified was not found or could not be opened.

Include files nested too deep: You used the Include directive in the 4NT.INI file, and attempted to nest include files more than three levels deep.

Infinite COPY or MOVE loop: You tried to COPY or MOVE a directory to one of its own subdirectories and used the /S switch, so the command would run forever. Correct the command and try again.

Insufficient disk space: COPY or MOVE ran out of room on the destination drive. Remove some files and retry the operation.

Invalid character value: You gave an invalid value for a character directive in the *4NT.INI* file.

Invalid choice value: You gave an invalid value for a "choice" directive (one that accepts a choice from a list, like "Yes" or "No") in the *4NT.INI* file.

Invalid color: You gave an invalid value for a color directive in the *4NT.INI* file.

Invalid count: The character repeat count for KEYSTACK is incorrect.

Invalid date: An invalid date was entered. Check the syntax and reenter.

Invalid directive name: 4NT can't recognize the name of a directive in your *4NT.INI* file.

Invalid drive: A bad or non-existent disk drive was specified.

Invalid key name: You tried to make an invalid key substitution in the *4NT.INI* file, or you used an invalid key name in a keystroke alias or command. Correct the error and retry the operation.

Invalid numeric value: You gave an invalid value for a numeric directive in the *4NT.INI* file.

Invalid parameter: 4NT didn't recognize a parameter. Check the syntax and spelling of the command you entered.

Invalid path: The specified path does not exist. Check the disk specification and/or spelling.

Invalid path or file name: You used an invalid path or filename in a directive in the *4NT.INI* file.

Invalid time: An invalid time was entered. Check the syntax and reenter.

Keystroke substitution table full: 4NT ran out of room to store keystroke substitutions entered in the *4NT.INI* file. Reduce the number of key substitutions or contact JP Software or your dealer for assistance.

Label not found: A GOTO or GOSUB referred to a non-existent label. Check your batch file.

Missing ENDTEXT: A TEXT command is missing a matching ENDTEXT. Check the batch file.

Missing GOSUB: 4NT cannot perform the RETURN command in a batch file. You tried to do a RETURN without a GOSUB, or your batch file has been corrupted.

Missing SETLOCAL: An ENDLOCAL was used without a matching SETLOCAL.

No aliases defined: You tried to display aliases but no aliases have been defined.

No closing quote: 4NT couldn't find a second matching back quote [`] or double-quote [**"**] on the command line.

No expression: The expression passed to the @EVAL variable function is empty. Correct the expression and retry the operation.

No shared memory found: The SHRALIAS command could not find any global alias list, history list, or directory history list to retain, because you executed the command from a session with local lists. Start 4NT with at least one global list, then invoke SHRALIAS.

No room for INI file name: 4NT does not have enough space to pass the name of your *4NT.INI* file to secondary shells; see **String area overflow** for more details. Any [Secondary] section in *4NT.INI* will be ignored in secondary shells until the problem is corrected and the system or session is restarted.

Not a directory: You tried to use the RD /S command with a parameter that is not a directory.

Not an alias: The specified alias is not in the alias list.

Not in environment: The specified variable is not in the environment.

Not ready: The specified device can't be accessed.

Not same device: This error usually appears in RENAME. You cannot rename a file to a different disk drive.

Out of memory: 4NT or Windows NT had insufficient memory to execute the last command. Try to free some memory by closing other sessions. If the error persists, contact JP Software for assistance.

Out of paper: Windows NT detected an out-of-paper condition on one of the printers. Check your printer and add paper if necessary.

Overflow: An arithmetic overflow occurred in the @EVAL variable function. Check the values being passed to @EVAL.   @EVAL can handle 16 digits to the left of the decimal point and 8 to the right.

Read error: Windows NT encountered a disk read error; usually caused by a bad or unformatted disk.   Also see **Data error** above.

Sector not found: Disk error, usually caused by a bad or unformatted disk.   Also see **Data error** above.

Seek error: Windows NT can't seek to the proper location on the disk.   This is generally caused by a bad disk or drive.   Also see **Data error** above.

Sharing violation: You tried to access a file in use by another program in a multitasking system or on a network.   Wait for the file to become available, or change your method of operation so that another program does not have the file open while you are trying to use it.

SHRALIAS already loaded: You used the SHRALIAS command to load *SHRALIAS.EXE*, but it was already loaded.   This message is informational and generally does not indicate an error condition.

SHRALIAS not loaded: You used the SHRALIAS /U command to unload *SHRALIAS.EXE*, but it was never loaded.   This message is informational and may not indicate an error condition.

String area overflow: 4NT ran out of room to store the text from string directives in the *4NT.INI* file.   Reduce the complexity of the *4NT.INI* file or contact JP Software for assistance.

Syntax error: A command or variable function was entered in an improper format.   Check the syntax and correct the error.

Too many open files: Windows NT has run out of file handles.

Unbalanced parentheses: The number of left and right parentheses did not match in an expression passed to the @EVAL variable function.   Correct the expression and retry the operation.

Unknown command: A command was entered that 4NT didn't recognize and couldn't find in the current search path.   Check the spelling or PATH specification.   You can handle unknown commands with the UNKNOWN_CMD alias (see ALIAS).

UNKNOWN_CMD loop: The UNKNOWN_CMD alias called itself more than ten times.   The alias probably contains an unknown command itself, and is stuck in an infinite loop.   Correct the alias.

Variable loop: A nested environment variable refers to itself, or variables are nested more than 16 deep.   Correct the error and retry the command.

Window title not found: The specified window does not exist.

Write error: Windows NT encountered a disk write error; usually caused by a bad or unformatted disk.   Also see **Data error** above.

Troubleshooting, Service, and Support

If you need help with 4NT, we encourage you to review our documentation and then contact us for assistance if required.

If you need help with **sales**, **ordering**, or **shipments** (including defective disks or other materials which were shipped to you), or with **brand codes**, please contact our **Sales and Customer Service** department. See Contacting JP Software for our email address, mail address, and telephone numbers. (The sales and customer service staff cannot assist you with technical problems. However, if you have multiple questions or are unsure of the nature of the problem, feel free to contact us for customer service; the staff will have a support technician contact you if your question turns out to require technical expertise.)

If you need **technical support** for 4NT, review the Technical Support information section, which tells you what we need to know to provide you with accurate and timely support. Then contact us via one of the methods described there. (The technical support staff cannot assist you with sales, ordering, replacement brand cards, or other administrative matters.)

Technical Support

Before You Contact Us

Before contacting us for support, please check this help file, the Reference Manual and other documentation for answers to your question. If you can't find what you need, try the Index. If you're having trouble getting 4NT to run properly, review the information on Error Messages, and look through the *README.DOC* file for any last-minute information.

If you do need to contact us for support, we can do a much better job of assisting you if you can give us some basic information, separate from your interpretations of or conclusions about the problem. The first four items listed below are essential for us to be able to understand and assist you with your problem:

» **What environment are you working in?** This includes the operating system version are you using, the version of the JP Software product involved, and related information such as network connections and the name and version number of any other software which appears to be involved in the problem. Use the VER /R command to determine the 4NT version and operating system version.

» **What exactly did you do?** A concise description of what steps you must take to make the problem appear is much more useful than a long analysis of what might be happening.

» **What did you expect to happen?** Tell us the result you expected from the command or operation in question, so that we understand what you are trying to do.

» **What actually happened?** At what point did the failure occur? If you saw an error message or other important or unusual information on the screen, what **exactly** did it say?

» **Briefly, what techniques did you use to try to resolve the problem?** What results did you get?

» **If the problem seems related to startup and configuration issues**, what are the contents of any startup files you use (such as *4START*, *4EXIT*, and the *4NT.INI* file), any batch files they call, and any alias or environment variable files they load?

» **Can you repeat the problem or does it occur randomly?** If it's random, does it seem related to the programs you're using when the problem occurs?

Electronic Support

Usually the best way to contact us for support is via CompuServe or the Internet. The most efficient method is to use our CompuServe support conference; if you do not have CompuServe access, contact us via Internet email. See Contacting JP Software for our addresses.

Whenever possible, we also read messages posted on the Usenet *comp.os.msdos.4dos* newsgroup, and in 4DOS conferences on the RIME, Ilink, and FidoNet BBS networks (these conferences are named for 4DOS, but carry messages related to all JP Software products). These areas offer valuable information and discussions with other users, but are not managed by JP Software, and are not official support channels. To be certain of a direct answer from our support staff use our CompuServe forum or Internet email, or contact us by telephone, fax, or mail.

A number of support resources are available from our web site listed above, including error message listings, documentation files, product histories, technical tips and discussions, other technical information, and links to other companies' sites. We update this information regularly, and we encourage you to check the Technical Support area of the web site to see if the information there will address any questions you have.

Technical support messages should be sent as standard ASCII text. Please do not transmit attached files, binary files, screen images, or any file over 10K bytes in size to any of our electronic technical support addresses unless asked to do so by our support staff.

Telephone Support

Technical support by telephone within the US and Canada is handled on a callback basis. To contact our support staff, call the US / Canada Support Line at any time and leave a short voice mail message describing your technical problem (this line can **not** be used for sales / customer service issues such as pricing, ordering, upgrades, or shipping problems). We check these messages regularly throughout the day and will return your call as quickly as possible. See Contacting JP Software for our phone numbers.

We generally return all technical support calls within 24 hours (weekends and holidays excluded), and most are returned much more quickly, usually on the same business day. If your problem is urgent and requires a faster response, please let us know and we will try to accommodate you. If you contact us by telephone and don't receive a reply within 24 hours, please try again. We probably tried to return your call and were unable to reach you.

If you are calling from outside the US and Canada, are not sure if your question requires technical support, need other assistance in addition to your technical questions, or find yourself playing "telephone tag" with our support staff, please call our main number listed above. Our office staff will assist you with all of your concerns, and have a technical support representative call you back if necessary.

If you have a problem with a batch file or complex alias, please contact us electronically if possible. Include a copy of the batch file or alias in question, preferably as part of the text of your message (not as an attachment). If you do not have electronic access, contact us by fax if possible. Problems of this type are usually very difficult to diagnose over the telephone because we cannot see the material you are working with. For longer batch files (over about 25 lines), do your best to reproduce the problem in a smaller test file.

If you need more in-depth assistance with the development of complex batch files or other procedures, please contact us for information on consulting services.

Contacting JP Software

You can contact JP Software at the following addresses and numbers. Our normal business hours are 8:30 AM to 5:00 PM weekdays, eastern US time (except holidays).

Address: JP Software Inc.
P.O. Box 1470
East Arlington, MA 02174
USA

Main number: (617) 646-3975 **

Fax: (617) 646-0904 **

Order Line: (800) 368-8777 (US / Canada, orders only)

Support Line: (617) 646-0798 ** (US / Canada only; **see Telephone Support before using this number**)

**** Please Note:** Our (617) area code will change to (781) effective September 1, 1997

Internet: Sales / Customer Service: **sales@jpsoft.com**

Technical Support: **support@jpsoft.com**

World Wide Web: **http://www.jpsoft.com/**

File downloads via FTP: For the simplest access to JP Software files use our web site. For direct FTP access connect to **ftp.std.com** and look in the **/vendors/jpsoft** directory.

CompuServe: Sales / Customer Service: **75020,244**

Technical Support and File Downloads: **GO JPSOFT** or **GO PCVENB**, section / library 10, User ID 75300,1215.

BBS Downloads: Channel 1 BBS, Boston, 617-349-1300 at 2,400 - 28,800 baud, no parity, 8 data bits, 1 stop bit.

File Systems and File Name Conventions

You may have dozens, hundreds, or thousands of files stored on your computer's disks. Your operating system is responsible for managing all of these files. In order to do so, it uses a unique name to locate each file in much the same way that the post office assigns a unique address to every residence.

The unique name of any file is composed of a **drive letter**, a directory **path**, and a **filename**. Each of these parts of the file's name is case insensitive; you can mix upper and lower case letters in any way you wish.

The topics below are roughly divided according to the different parts of a file name, and cover the file system structure and naming conventions:

Drives and Volumes

File Systems

Directories and Subdirectories

File Names

File Attributes and Time Stamps

Drives and Volumes

A **drive letter** designates which drive contains the file. In a file's full name, the drive letter is followed by a colon. Drive letters **A:** and **B:** are normally reserved for the floppy disk drives.

Normally, drive **C:** is the first (or only) hard disk drive. Most current operating systems can divide a large hard disk into multiple logical drives or **volumes** that are usually called **C:**, **D:**, **E:**, etc. Network systems (LANs) give additional drive letters to sections of the network file server drives.

Most recent systems also include a CD-ROM drive. The CD-ROM is also assigned a drive letter (or several letters, for CD-ROM changers), typically using letters beyond that used by the last hard disk in the system, but before any network drives. Some systems may have "RAM disks" (sometimes called "virtual disks"), which are areas of memory set aside by software (a "RAM disk driver") for use as fast but temporary storage. Like CD-ROM drives, RAM disks are usually assigned drive letters beyond the last hard disk in the system, but before network drives.

For example, on a system with a large hard disk you might have **A:** and **B:** as floppy drives, **C:**, **D:**, and **E:** as parts of the hard disk, **F:** as a CD-ROM drive, **G:** as a RAM disk, and **H:** and **I:** as network drives.

Each volume is formatted under a particular file system; see File Systems for details. Additional information about disk files and directories is available under Directories and Subdirectories, File Names, and File Attributes and Time Stamps.

File Systems

Each disk volume is organized according to a **file system**. The file system determines how files are named and how they are organized on the disk.

As hard disk technology and operating systems have evolved, new file systems have been invented to support longer file names, larger drives, and higher disk performance. Several different and incompatible schemes have evolved. Which file systems you can use depends on which operating system you are using, and how the operating system and your hard disk are configured.

The operating systems under which 4NT runs can support four standard file systems: FAT, VFAT, HPFS, and NTFS. See File Names for details on the rules for naming files under each file system.

» The **FAT File System** is the traditional file system used by all versions of DOS. Its name comes from the **F**ile **A**llocation **T**able DOS uses to keep track of the space allocated to each file. Windows 95, Windows NT, and OS/2 also support the FAT file system.

» The **VFAT File System** is an extension of the FAT file system available in Windows NT and Windows 95, including DOS and 4DOS sessions run from the Windows 95 desktop. This system maintains additional information about files on FAT drives, including long filenames.

Other operating systems (OS/2 and earlier versions of DOS) can access files on VFAT drives, but will not be able to access long filenames or other information which is added by the VFAT file system.

Throughout this manual, the term "LFN file system" is used to describe the VFAT system (LFN stands for **L**ong **F**ile **N**ame).

» The **High Performance File System** or **HPFS** is a file system provided with all versions of OS/2, and is also supported in Windows NT version 3.51 and below. It supports long file names, and offers higher performance and better support for large drives than the FAT or VFAT system. It also supports "extended attributes" to retain additional information about your files.

DOS sessions running under OS/2 can access files on HPFS drives if the files have short, FAT-compatible names. Other operating systems (DOS, Windows 95, and Windows NT 4.0 and above) can not access files on HPFS drives.

» The **Windows NT File System** or **NTFS** is a file system provided with all versions of Windows NT. It supports long file names, and offers higher performance and better support for large drives than the FAT or VFAT system.

DOS programs running under Windows NT can access files on NTFS drives if the files have short, FAT-compatible names. Other operating systems (DOS, Windows 95, and OS/2) can not access files on NTFS drives.

Additional file systems may be installed under some operating systems to support CD-ROM or network drives. The file system type (FAT / VFAT, HPFS, or NTFS) is determined when a hard disk volume is formatted and applies to the entire volume. For example, under Windows NT you might have a 2 GB hard disk divided into four 500 MB volumes, with the first three volumes (C:, D:, and E:) formatted for the FAT or VFAT file system, and the fourth formatted for NTFS.

4NT supports any standard file system installed under your operating system. If your operating system can access files on a particular drive, then 4NT will be able to access those files as well.

Additional information about disk files and directories is available under Drives and Volumes, Directories and Subdirectories, File Names, and File Attributes and Time Stamps.

Network File Systems

A network file system allows you to access files stored on another computer on a network, rather than on your own system.   4NT supports all network file systems which are compatible with the underlying operating system.

File and directory names for network file systems depend on both the "server" software running on the system that has the files on it, and the "client" software running on your computer to connect it to the network.   However, they usually follow the rules described under File Names.

Most network software "maps" unused drive letters on your system to specific locations on the network, and you can then treat the drive as if it were physically part of your local computer.

Some networks also support the **Universal Naming Convention**, which provides a common method for accessing files on a network drive without using a "mapped" drive letter.   Names specified this way are called UNC names.   They typically appear as **\\server\volume\path\filename**, where **server** is the name of the network server where the files reside, **volume** is the name of a disk volume on that server, and the **path\filename** portion is a directory name and file name which follow the conventions described in Directories and Subdirectories.   4NT supports UNC filenames, and also allows you to use UNC directory names when changing directories (see Directory Navigation for more details).

When you use a network file system, remember that the naming conventions for files on the network may not match those on your local system.   For example, your local system may support long filenames while the network server or client software does not, or vice versa.   4NT will usually handle whatever naming conventions are supported by your network software, as long as the network software accurately reports the types of names it can handle.

In rare cases, 4NT may not be able to report correct statistics on network drives (such as the number of bytes free on a drive).   This is usually because the network file system does not provide complete or accurate information.

Directories and Subdirectories

A file system is a method of organizing all of the files on an entire disk or hard disk volume. **Directories** are used to divide the files on a disk into logical groups that are easy to work with. Their purpose is similar to the use of file drawers to contain groups of hanging folders, hanging folders to contain smaller manila folders, and so on. **Directories** are also sometimes referred to as **folders**.

Every drive has a **root** or base directory, and many have one or more **subdirectories**. Subdirectories can also have subdirectories, extending in a branching **tree** structure from the root directory. The collection of all directories on a drive is often called the **directory tree**, and a portion of the tree is sometimes called a **subtree**. The terms **directory** and **subdirectory** are typically used interchangeably to mean a single subdirectory within this tree structure.

Subdirectory names follow the same rules as file names (see File Names).

The drive and subdirectory portion of a file's name are collectively called the file's **path**. For example, the file name *C:\DIR1\DIR2\MYFILE.DAT* says to look for the file *MYFILE.DAT* in the subdirectory *DIR2* which is part of the subdirectory *DIR1* which is on drive C. The path for *MYFILE.DAT* is *C:\DIR1\DIR2*. The backslashes between subdirectory names are required. On NTFS, and LFN volumes the path and file name must each be 255 characters or less in length, and in addition the total length of the path and file name together cannot exceed 260 characters.

The operating system and command processor remember both a **current** or **default drive** for your system as a whole, and a **current** or **default directory** for every drive in your system. Whenever a program tries to create or access a file without specifying the file's path, the operating system uses the current drive (if no other drive is specified) and the current directory (if no other directory path is specified).

The root directory is named using the drive letter and a single backslash. For example, *D:\* refers to the root directory of drive *D:*. Using a drive letter with no directory name at all refers to the current directory on the specified drive. For example, *E:README.DOC* refers to the file *README.DOC* in the current directory on drive *E:*, whereas *E:\README.DOC* refers to the file *README.DOC* in the root directory on drive *E:*.

There are also two special subdirectory names that are useful in many situations: a single period by itself [**.**] means "the current default directory." Two periods together [**..**] means "the directory which contains the current default directory" (often referred to as the **parent directory**). These special names can be used wherever a full directory name can be used. 4NT allows you to use additional periods to specify directories further "up" the tree (see Extended Parent Directory Names).

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, File Names, and File Attributes and Time Stamps.

File Names

Under the FAT file system, the filename consists of a **base name** of 1 to 8 characters plus an optional **extension** composed of a period plus 1 to 3 more characters. Traditional FAT filenames with an 8-character name and a 3-character extension are sometimes referred to as short filenames (SFNs) to distinguish them from long filenames (LFNs).

You can use alphabetic and numeric characters plus the punctuation marks **! # $ % & ' ( ) - @ ^ _ ` { }** and **~** in both the base name and the extension of a FAT filename. Because the exclamation point [**!**], percent sign [**%**], caret [**^**], at sign [**@**], parentheses [**()**], and back-quote [**`**] also have other meanings to 4NT, it is best to avoid using them in filenames.

The LFN, HPFS, and NTFS file systems allow file names with a maximum of 255 characters, including spaces and other characters that are not allowed in a FAT system file name, but excluding some punctuation characters which are allowed in FAT file names. See your operating system documentation for details on the characters allowed. If you use file names which contain semicolons [**;**], see Wildcards for details on avoiding problems with interpretation of those file names under 4NT.

NTFS, HPFS, and LFN file names are stored and displayed exactly as you entered them, and are not automatically shifted to upper or lower case. For example, you could create a file called *MYFILE*, *myfile*, or *MyFile*, and each name would be stored in the directory just as you entered it. However, case is ignored when looking for filenames, so you cannot have two files whose names differ only in case (*i.e.*, the three names given above would all refer to the same file). This behavior is sometimes described as "case-retentive but not case-sensitive" because the case information is retained, but does not affect access to the files.

Files stored on NTFS, HPFS, and LFN volumes often have "FAT-compatible" names: names which contain only those characters legal on a FAT volume, and which meet the 8-character name / 3-character extension limits. Programs which cannot handle long names (for example, Windows 3 programs accessing an NTFS drive under Windows NT) generally can access files by using FAT-compatible names.

If an NTFS, HPFS, or LFN-compatible file name includes spaces or other characters that would not be allowed in a FAT name, you **must** place double quotes around the name. For example, suppose you have a file named *LET3* on a FAT volume, and you want to copy it to the *LETTERS* directory on drive F:, an HPFS partition, and give it the name *Letter To Sara*. To do so, use either of these commands:

```
      [c:\wp] copy let3 f:\LETTERS\"Letter To Sara"
      [c:\wp] copy let3 "f:\LETTERS\Letter To Sara"
```

The NTFS, HPFS, and LFN file systems do not explicitly define an "extension" for file names which are not FAT-compatible. However, by convention, all characters after the last period in the file name are treated as the extension. For example, the file name *"Letter to Sara"* has no extension, whereas the name *"Letter.to.Sara"* has the extension *Sara*.

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, Directories and Subdirectories, and File Attributes and Time Stamps.

File Attributes and Time Stamps

Each file also has **attributes**, and one or more **time stamps**. Attributes define characteristics of the file which may be useful to the operating system, to you, or to an application program. Time stamps can record when the file was created, last modified, or last accessed. Most 4NT file processing commands allow you to select files for processing based on their attributes and/or time stamp(s).

Each file on your system has four standard attributes. Every time a program modifies a file, the operating system sets the **Archive** attribute, which signals that the file has been modified since it was last backed up. This attribute can be used by 4NT to determine which files to COPY, and by backup programs to determine which files to back up. When the **Read-only** attribute is set, the file can't be changed or erased accidentally; this can be used to help protect important files from damage. The **Hidden** and **System** attributes prevent the file from appearing in normal directory listings. (Two additional attributes, **Directory** and **Volume label**, are also available. These attributes are controlled by the operating system, and are not modified directly by 4NT.)

Attributes can be set and viewed with the ATTRIB command. The DIR command also has options to select filenames to view based on their attributes, to view the attributes themselves, and to view information about normally "invisible" hidden and system files.

When a file is created, and every time it is modified, the operating system records the system time and date in a **time stamp** in the file's directory entry. Several 4NT commands and variable functions, and many backup and utility programs, use this time stamp to determine the relative ages of files.

On FAT volumes, only the single time stamp described above is available. Files on NTFS, HPFS, and LFN volumes have three sets of time and date stamps. The operating system records when each file was created, when it was last written or modified, and when it was last accessed. The "last write" time stamp matches the single time stamp used on traditional FAT volumes.

Several 4NT commands and variable functions let you specify which set of time and date stamps you want to view or work with on NTFS, HPFS, and LFN volumes. These commands and functions use the letter "c" to refer to the creation time stamp, "w" for the last write time stamp, and "a" for the last access time stamp. Note that LFN volumes under Windows NT store a date but no time in the "last access" time stamp; on these drives the time of last access will always be 00:00.

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, Directories and Subdirectories, and File Names.

Miscellaneous Reference Information

Colors and Color Names

Keys and Key Names

Popup Windows

Executable Files and File Searches

Colors and Color Names

You can use color names in several of the directives in the *.INI* file (see Color Directives) and in many commands.   The general form of a color name is:

[BRIght] *fg* ON [BRIght] *bg*

where **fg** is the foreground or text color, and **bg** is the background color.

The available colors are:

```
Black       Blue        Green       Red
Magenta     Cyan        Yellow      White
```

Color names and the word BRIght may be shortened to the first 3 letters.

You can also specify colors by number instead of by name.   The numbers are most useful in potentially long *.INI* file directives like ColorDir, where using color names may take too much space.   The following numbers are recognized:

```
0 - Black         8 - Gray (bright black)
1 - Blue          9 - Bright blue
2 - Green         10 - Bright green
3 - Cyan          11 - Bright cyan
4 - Red           12 - Bright red
5 - Magenta       13 - Bright magenta
6 - Yellow        14 - Bright yellow
7 - White         15 - Bright white
```

Use one number to substitute for the **[BRIght] fg** portion of the color name, and a second to substitute for the **[BRIght] bg** portion.   For example, instead of **bright cyan on blue** you could use **11 on 1** to save space in a ColorDir specification.

Color Errors

A standard color specification allows sixteen foreground and sixteen background colors.   However, most video adapters and monitors do not provide true renditions of certain colors.   For example, most users see normal "yellow" as brown, and bright yellow as yellow; many also see normal red as red, and "bright red" as pink.   These problems are inherent in the monitor, video adapter, and driver software.   They cannot be corrected using 4NT color specifications.

Keys and Key Names

Key names are used to define keystroke aliases, and in several *4NT.INI* directives (see Key Mapping Directives). The format of a key name is the same in both uses:

```
[Prefix-]Keyname
```

The key prefix can be left out, or it can be any one of the following:

```
Alt   followed by A - Z, 0 - 9, F1 - F12, or Bksp
Ctrl  followed by A - Z, F1 - F12, Tab, Bksp, Enter, Left, Right, Home,
      End, PgUp, PgDn, Ins, or Del
Shift followed by F1 - F12 or Tab.
```

The possible key names are:

```
A - Z        Enter        PgDn
0 - 9        Up           Home
F1 - F12     Down         End
Esc          Left         Ins
Bksp         Right        Del
Tab          PgUp
```

All key names must be spelled as shown. Alphabetic keys can be specified in upper or lower case. You cannot specify a punctuation key.

The prefix and key name must be separated by a dash [**-**]. For example:

```
Alt-F10     This is okay
Alt F10     The space will cause an error
```

If you prefer, you can use a numeric value instead of a key name. Use the ASCII code for an ASCII, extended ASCII, or control character. Use the scan code preceded by an at sign [**@**] for extended key codes like **F1** or the cursor keys. For example, use 13 for **Enter**, or @59 for **F1**. In general, you will find it easier to use the names described above rather than key numbers.

Some keys are intercepted by Windows NT and are not passed on to 4NT. For example, **Ctrl-S** pauses screen output temporarily, and **Ctrl-Esc** pops up the Windows NT window list. Keys which are intercepted by Windows NT generally cannot be assigned to aliases or with key mapping directives, because 4NT never receives these keystrokes and therefore cannot act on them.

You also may not be able to use certain keys if your keyboard is not 100% IBM-compatible or your keyboard driver does not support them. For example, on some systems the **F11** and **F12** keys are not recognized; others may not support unusual combinations like **Ctrl-Tab**. These problems are rare; when they do occur, they are usually due to Windows NT.

Popup Windows

Several features of 4NT display popup windows. A popup window may be used to display filenames, recently-executed commands, recently-used directories, the results of an extended directory search, or a list created by the SELECT command or the @SELECT internal function.

Popup windows always display a list of choices and a cursor bar. You can move the cursor bar inside the window until you find the choice that you wish to make, then press the **Enter** key to select that item.

Navigation inside any popup window follows the conventions described below. Additional information on each specific type of popup window is provided when that window is introduced.

You can control the color, position and size of most popup windows from the Command Line 2 page of the OPTION dialogs. You can also control these features with directives in the *.INI* file, including PopupWinLeft, PopupWinTop, PopupWinWidth, and PopupWinHeight, and PopupWinColors. A few popup windows (*e.g.*, the extended directory search window) have their own specific *.INI* directives, and corresponding separate choices in the OPTION dialogs. You can also change the keys used in most popup windows with key mapping directives in the *.INI* file.

Once a window is open, you can use these navigation keys to find the selection you wish to make:

	Move the selection bar up one line.
↓	Move the selection bar down one line.
←	Scroll the display left 4 columns.
→	Scroll the display right 4 columns.
PgUp	Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the list.
Ctrl-PgDn or **End**	Go to the end of the list.
Esc	Close the window without making a selection.
Enter	Select the current item and close the window.

In addition to scrolling through a popup window, you can search the list using character matching. If you press a character, the cursor bar will move to the next entry that begins with that character. If you type multiple characters, the cursor will move to the entry that begins with the search string entered to that point (you can enter a search string up to 32 characters long). If no entry matches the character or string that you have typed, the command processor beeps and does not move the cursor bar. To reset the search string, press **Backspace**.

You can change the keys used in popup windows with key mapping directives in the *.INI* file.

Executable Files and File Searches

Once 4NT knows that it is supposed to run an external command, it tries to find an executable file (one with a *.COM* or *.EXE* extension) whose name matches the command name. It runs the executable file if it finds one.

If 4NT cannot find an executable program to run, it next looks for a batch file (a file with one or more commands in it) whose name matches the command name. 4NT looks first for a *.BTM* file, then for a *.CMD* file, then for a *.BAT* file, and finally for a *.REX* file. See *.BAT, .CMD*, and *.BTM* Files for more information on these different types of batch files. If 4NT finds such a file, it then reads each line in the file as a new command.

If the search for a batch file fails, 4NT checks to see if the command name matches the name of a file with an extension that is associated with a specific application (for example, if you have associated *.DOC* with your editor or word processor, and you type the name of a *.DOC* file). If a match is found, 4NT runs the program you specified when the association was defined.

In searching for the application associated with a file, 4NT will first examine any executable extensions you have defined to associate a file extension with a specific program to process that type of file. It then checks file associations defined in the Windows registry; see Windows File Associations for complete details.

4NT first searches for an executable program, a batch file, and a file with an executable extension (or Windows NT file association) in the current directory. If the command name doesn't match a *.COM*, *.EXE*, *.BTM*, *.BAT* or *.CMD* file or an executable extension in the current directory, 4NT repeats its search in every directory in your **search path**.

The search path is a list of directories that 4NT (and some applications) search for executable files. For example, if you wanted 4NT to search the root directory of the C: drive, the \NT subdirectory on the C: drive, and the \UTIL directory on the D: drive for executable files, your search path would look like this:

```
PATH=C:\;C:\NT;C:\UTIL
```

Notice that the directory names in the search path are separated by semicolons.

You can create or view the search path with the PATH command. You can use the ESET command to edit the path. Many programs also use the search path to find their own files. The search path is stored in the environment with the name PATH.

Remember, 4NT always looks for an executable file or a file with an executable extension or Windows file association in the current subdirectory, then in each directory in the search path. (You can change the search order so the current directory is not searched first; see the PATH command for details.)

If you include an extension as part of the command name, 4NT only searches for a file with that extension. Similarly, if you include a path as part of the command name, the command processor will look only in the directory you specified, and ignore the usual search of the current directory and the PATH.

The following table sums up the possible search options (the term "standard search" refers to the search of the current directory and each directory in the search path):

Command	4NT Search Sequence
WP	Standard search for any executable file whose base name is *WP*.
WP.EXE	Standard search for *WP.EXE*; will not find files with other extensions.

C:\WP7\WP	Looks in the *C:\WP7* directory for any executable file whose base name is *WP*. Does not check the standard search directories.
C:\WP7\WP.EXE	Looks only for the file *C:\WP7\WP.EXE*.
LAB.DOC	Standard search for *LAB.DOC*, if *.DOC* is defined as an executable extension (or Windows NT file association). Runs the associated application if the file is found.
C:\LI\LAB.DOC	Looks only for the file *C:\LI\LAB.DOC*, and only if *.DOC* is defined as an executable extension (or Windows NT file association). Runs the associated application if the file is found.

If 4NT cannot find an executable file, batch program, or a file with an executable extension (or Windows NT file association) in the current directory or any directory in the search path, it looks for an alias called UNKNOWN_CMD (see the ALIAS command for details). If you have defined an alias with that name, it is executed (this allows you to control error handling for unknown commands). Otherwise, 4NT displays an "Unknown command" error message and waits for your next instruction.

Windows File Associations

Windows NT includes the ability to associate file extensions with specific applications; this feature is sometimes called "file associations". For example, when you install Microsoft Word, it creates an association between files with a *.DOC* or *.DOT* extension and the Word program file, *WINWORD.EXE*.

4NT includes a similar feature, called executable extensions, which allows you to set environment variables which associate a file extension with a particular application.

There are two sets of file associations; both are stored in the Windows NT registry. Those in the first group (designed to replace the Windows 3.x *WIN.INI* file) are "direct" — they simply list the extension and the application name, associating the extension directly with the application. Those in the second group are "indirect" — they associate an extension with a "file type," and separately specify attributes of files of that type, including the command to execute in order to open such a file.

When 4NT starts, it retrieves the direct file associations from Windows, and treats them like 4NT executable extensions. To disable loading of these direct associations see the Startup page of the OPTION and the LoadAssociations directive in *4NT.INI*. The indirect associations are accessed through built-in Windows features (see below), and can not be disabled.

When you attempt to execute a program from the command line or a batch file, 4NT first searches its list of executable extensions, including the standard extensions (*.COM*, *.EXE*, *.BTM*, etc.). It then checks 4NT executable extensions, followed by direct file associations inherited from Windows. If all of these tests fail, 4NT passes the command name to Windows to see if Windows can find an indirect association for it. Each of these tests is done in all of the standard search directories described under Executable Files and File Searches.

See Using Windows File Associations for additional details on how to control Windows file associations in 4NT, including dealing with conflicts between direct and indirect associations.

ASCII and Key Codes

For ASCII and key code reference tables, see:

> ASCII Tables
>
> Key Codes and Scan Codes Tables

The remainder of this section gives a detailed explanation of character sets, ASCII, and key codes. If you are troubleshooting a keyboard or character display problem be sure to read all of the explanation below before referring to the tables.

The translation of a key you type on the keyboard to a displayed character on the screen depends on several related aspects of character handling. A complete discussion of these topics is well beyond the scope of this document. However, a basic picture of the steps in the keystroke and character translation process will help you understand how characters are processed in your system, and why they occasionally may not come out the way you expect.

Internally, computers use numbers to represent the keys you press and the characters displayed on the screen. To display the text that you type, your computer and operating system require five pieces of information:

» The numeric **key code** for the physical key you pressed.

» The specific character that key code represents based on your current keyboard layout or **country setting**.

» The **character set** currently in use on your system (see below).

» The international **code page** in use for that character set.

» The **display font** used to display the character.

The numeric **key code** is determined by your physical hardware including the language that your keyboard is produced for. The **character set** is usually determined by the operating system. These items typically are not under your control. However, most systems do allow you to control the keyboard **country setting,** the **code page**, and the **display font**.

For an explanation of how key codes work, see the Key Codes and Scan Codes Explanation. For a list of key codes and scan codes for keys on the standard U.S. keyboard see the Key Codes and Scan Codes Tables.

If the key codes produced by your keyboard, the code page, and the font you choose are not fully compatible with each other, the characters displayed on the screen will not match what you type. The differences are likely to appear in line-drawing characters, "international" (non-English) characters, and special symbols, not in commonly-used U.S. English alphabetic, numeric, or punctuation characters.

Most systems use a "single-byte" character set for keyboard and screen display. These sets define 256 characters or symbols, with a numeric representation for each. ("Double-byte" character sets, with up to 65,536 characters each, are used for languages with more than 256 symbols, and for some multi-lingual systems.) Most PC single-byte character sets are based on a code called ASCII, the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. For a complete list of ASCII codes see the ASCII Tables.

The original ASCII code was defined over 30 years ago for use in mainframe and minicomputer systems, and has 128 character values. These include the upper and lower case letters, numerals, and punctuation marks used in U.S. English, plus non-printing control codes (which can be entered on most keyboards by pressing the **Ctrl** key plus another character, or by pressing certain special keys like **Tab**,

Enter, **Backspace**, and **Esc**). However, ASCII is not a complete character set, because it defines only 128 of the required 256 symbols.

IBM, in its original PC, created a complete 256-character set (called the Original Equipment Manufacturer or "OEM" character set) by defining an additional 128 **extended ASCII** codes for math symbols, "international" characters, the characters used to draw boxes and lines, and some miscellaneous symbols.

Some operating systems support other character sets; in particular, Windows uses the ANSI character set internally to store and display text, even though other parts of the system (*e.g.* the file system which stores file names on disk) use IBM's OEM character set. The ANSI character set is identical to the OEM character set for U.S. English printed characters, but may vary for "international" characters not used in U.S. English. In most cases, Windows automatically translates characters from one set to another as needed, but problems can sometimes result in errors in displayed text (*e.g.*, differences between the appearance of accented characters in filenames in Windows and DOS applications).

See your operating system documentation for more information about character sets, code pages, and country and language support. Refer to your operating system and/or font documentation for details on the full character set available in any particular font.

The tables in this help file are based on U.S. English conventions. Your system may differ if it is configured for a different country or language. See your operating system documentation for more information about country and language support.

ASCII Tables

These tables shows the 256-character ASCII character set for U.S. English systems. Most of the first 128 characters will be the same on non-U.S. systems. Characters 1 through 31 and 128 through 255 are likely to vary in appearance between fonts, and may look different if your system is not configured for U.S. English.

For more details on ASCII, character sets, and key codes, see the general information topic on ASCII and Key Codes.

The tables below are primarily in the MS Line Draw font, as this font gives the best representation of standard ASCII characters. The Control Characters portion of the table also uses the Symbol and WingDings fonts to display the best possible approximation of the standard screen characters shown in common ASCII charts. If you do not have all of these fonts on your system the characters may not appear correctly.

Certain characters are shown as blank in the tables below: Characters 000 and 255 are blank because they are normally shown that way in ASCII charts; characters 002, 008, 010 - 014, and 023 are blank because the standard symbols for these characters have no representation in typical Windows fonts, so they cannot be shown here.

Control Characters (0 - 31)

Dec	Hex	Char	Name	Ctrl	Dec	Hex	Char	Name	Ctrl
000	00		NUL	^@	016	10	Ø	DLE	^P
001	01	J	SOH	^A	017	11	×	DC1	^Q
002	02		STX	^B	018	12	ô	DC2	^R
003	03		ETX	^C	019	13	!	DC3	^S
004	04	◆	EOT	^D	020	14	¶	DC4	^T
005	05	♣	ENQ	^E	021	15	§	NAK	^U
006	06	♠	ACK	^F	022	16	n	SYN	^V
007	07	•	BEL	^G	023	17		ETB	^W
008	08		BS	^H	024	18		CAN	^X
009	09	m	HT	^I	025	19	↓	EM	^Y
010	0A		LF	^J	026	1A	→	SUB	^Z
011	0B		VT	^K	027	1B	←	ESC	^[
012	0C		FF	^L	028	1C	_	FS	^\
013	0D		CR	^M	029	1D	↔	GS	^]
014	0E		SO	^N	030	1E	Ù	RS	^^
015	0F	R	SI	^O	031	1F	Ú	US	^_

Standard ASCII Printing Characters (32 - 127)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
032	20	Space	064	40	@	096	60	`
033	21	!	065	41	A	097	61	a
034	22	"	066	42	B	098	62	b
035	23	#	067	43	C	099	63	c
036	24	$	068	44	D	100	64	d
037	25	%	069	45	E	101	65	e
038	26	&	070	46	F	102	66	f

039	27	'	071	47	G	103	67	g
040	28	(	072	48	H	104	68	h
041	29	)	073	49	I	105	69	i
042	2A	*	074	4A	J	106	6A	j
043	2B	+	075	4B	K	107	6B	k
044	2C	,	076	4C	L	108	6C	l
045	2D	-	077	4D	M	109	6D	m
046	2E	.	078	4E	N	110	6E	n
047	2F	/	079	4F	O	111	6F	o
048	30	0	080	50	P	112	70	p
049	31	1	081	51	Q	113	71	q
050	32	2	082	52	R	114	72	r
051	33	3	083	53	S	115	73	s
052	34	4	084	54	T	116	74	t
053	35	5	085	55	U	117	75	u
054	36	6	086	56	V	118	76	v
055	37	7	087	57	W	119	77	w
056	38	8	088	58	X	120	78	x
057	39	9	089	59	Y	121	79	y
058	3A	:	090	5A	Z	122	7A	z
059	3B	;	091	5B	[	123	7B	{
060	3C	<	092	5C	\	124	7C	|
061	3D	=	093	5D	]	125	7D	}
062	3E	>	094	5E	^	126	7E	~
063	3F	?	095	5F	_	127	7F	

Extended ASCII Characters (128 - 255)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	€	171	AB	«	214	D6	Ö
129	81	▯	172	AC	¬	215	D7	×
130	82	‚	173	AD		216	D8	Ø
131	83	ƒ	174	AE	®	217	D9	Ù
132	84	„	175	AF	¯	218	DA	Ú
133	85	…	176	B0	°	219	DB	Û
134	86	†	177	B1	±	220	DC	Ü
135	87	‡	178	B2	²	221	DD	Ý
136	88	ˆ	179	B3	³	222	DE	Þ
137	89	‰	180	B4	´	223	DF	ß
138	8A	Š	181	B5	µ	224	E0	à
139	8B	‹	182	B6	¶	225	E1	á
140	8C	Œ	183	B7	·	226	E2	â
141	8D	▯	184	B8	¸	227	E3	ã
142	8E	Ž	185	B9	¹	228	E4	ä
143	8F	▯	186	BA	º	229	E5	å
144	90	▯	187	BB	»	230	E6	æ
145	91	'	188	BC	¼	231	E7	ç
146	92	'	189	BD	½	232	E8	è
147	93	"	190	BE	¾	233	E9	é
148	94	"	191	BF	¿	234	EA	ê
149	95	•	192	C0	À	235	EB	ë
150	96	–	193	C1	Á	236	EC	ì
151	97	—	194	C2	Â	237	ED	í
152	98	˜	195	C3	Ã	238	EE	î

153	99	™	196	C4	Ä	239	EF	ï
154	9A	š	197	C5	Å	240	F0	ð
155	9B	›	198	C6	Æ	241	F1	ñ
156	9C	œ	199	C7	Ç	242	F2	ò
157	9D	▯	200	C8	È	243	F3	ó
158	9E	ž	201	C9	É	244	F4	ô
159	9F	Ÿ	202	CA	Ê	245	F5	õ
160	A0		203	CB	Ë	246	F6	ö
161	A1	¡	204	CC	Ì	247	F7	÷
162	A2	¢	205	CD	Í	248	F8	ø
163	A3	£	206	CE	Î	249	F9	ù
164	A4	¤	207	CF	Ï	250	FA	ú
165	A5	¥	208	D0	Ð	251	FB	û
166	A6	¦	209	D1	Ñ	252	FC	ü
167	A7	§	210	D2	Ò	253	FD	ý
168	A8	¨	211	D3	Ó	254	FE	þ
169	A9	©	212	D4	Ô	255	FF	
170	AA	ª	213	D5	Õ			

Key Codes and Scan Codes Tables

(For more details on key codes, scan codes, and ASCII see the general information on ASCII and Key Codes, and the Key Codes and Scan Codes Explanation.)

The following table lists all of the keys on the 101-key "enhanced" U.S. keyboard. Many non-U.S. keyboards are similar, but will not be exactly the same. The keys are arranged roughly in scan code order, which is generally left to right, moving from the top of the keyboard to the bottom.

Column 1 shows the key's keycap symbol or name. Columns 2 and 3 show the scan code, and the ASCII code if the key is unshifted. Columns 4 and 5 contain the codes for the shifted key. Columns 6 and 7 show the codes for Ctrl plus the key. The last column contains the scan code for Alt plus the key (Alt keystrokes have no ASCII code and always generate an ASCII code of 0, which is not shown).

Key names prefaced by **np** are on the numeric keypad. Those prefaced by **cp** are on the cursor keypad between the main typing keys and the number keypad. The numeric keypad values are valid if Num Lock is turned off. If you need to specify a number key from the numeric keypad when Num Lock is on, use the scan code shown for the keypad and the ASCII code shown for the corresponding typewriter key. For example, the keypad "7" has a scan code of 71 (the np Home scan code) and an ASCII code of 54 (the ASCII code for "7").

The chart is blank for key combinations that do not have scan codes or ASCII codes, like **Ctrl-1** or **Alt-PgUp**.

Top Keyboard Row

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt Scan Code
Esc	1	27	1	27	1	27	1
1 !	2	49	2	33			120
2 @	3	50	3	64	3	0	121
3 #	4	51	4	35			122
4 $	5	52	5	36			123
5 %	6	53	6	37			124

6 ^	7	54	7	94	7	30	125
7 &	8	55	8	38			126
8 *	9	56	9	42			127
9 (	10	57	10	40			128
0)	11	48	11	41			129
- _	12	45	12	95	12	31	130
= +	13	61	13	43			131
Backspace	14	8	14	8	14	127	14

Second Keyboard Row

			Shift	Shift	Ctrl	Ctrl	Alt
Key Cap Symbol	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code
Tab	15	9	15	0	148	0	165
Q	16	113	16	81	16	17	16
W	17	119	17	87	17	23	17
E	18	101	18	69	18	5	18
R	19	114	19	82	19	18	19
T	20	116	20	84	20	20	20
Y	21	121	21	89	21	25	21
U	22	117	22	85	22	21	22
I	23	105	23	73	23	9	23
O	24	111	24	79	24	15	24
P	25	112	25	80	25	16	25
[{	26	91	26	123	26	27	26
] }	27	93	27	125	27	29	27
Enter	28	13	28	13	28	10	28

Third Keyboard Row

			Shift	Shift	Ctrl	Ctrl	Alt
Key Cap Symbol	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code
A	30	97	30	65	30	1	30
S	31	115	31	83	31	19	31
D	32	100	32	68	32	4	32
F	33	102	33	70	33	6	33
G	34	103	34	71	34	7	34
H	35	104	35	72	35	8	35
J	36	106	36	74	36	10	36
K	37	107	37	75	37	11	37
L	38	108	38	76	38	12	38
; :	39	59	39	58			39
' "	40	39	40	34			40
` ~	41	96	41	126			41
\ |	43	92	43	124	43	28	43

Bottom Keyboard Row

			Shift	Shift	Ctrl	Ctrl	Alt

Key Cap Symbol	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code	ASCII Code	Scan Code
Z	44	122	44	90	44	26	44
X	45	120	45	88	45	24	45
C	46	99	46	67	46	3	46
V	47	118	47	86	47	22	47
B	48	98	48	66	48	2	48
N	49	110	49	78	49	14	49
M	50	109	50	77	50	13	50
, <	51	44	51	60			51
. >	52	46	52	62			52
/ ?	53	47	53	63			53
Space	57	32	57	32	57	32	57

Function Keys

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt Scan Code
F1	59	0	84	0	94	0	104
F2	60	0	85	0	95	0	105
F3	61	0	86	0	96	0	106
F4	62	0	87	0	97	0	107
F5	63	0	88	0	98	0	108
F6	64	0	89	0	99	0	109
F7	65	0	90	0	100	0	110
F8	66	0	91	0	101	0	111
F9	67	0	92	0	102	0	112
F10	68	0	93	0	103	0	113
F11	133	0	135	0	137	0	139
F12	134	0	136	0	138	0	140

Numeric Key Pad

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt Scan Code
np *	55	42	55	42	150	0	55
np Home	71	0	71	55	119	0	
np Up	72	0	72	56	141	0	
np PgUp	73	0	73	57	132	0	
np Minus	74	45	74	45	142	0	74
np Left	75	0	75	52	115	0	
np 5	76	0	76	53	143	0	
np Right	77	0	77	54	116	0	
np Plus	78	43	78	43	144	0	78
np End	79	0	79	49	117	0	
np Down	80	0	80	50	145	0	
np PgDn	81	0	81	51	118	0	
np Ins	82	0	82	48	146	0	
np Del	83	0	83	46	147	0	

np /	224	47	224	47	149	0	164
np Enter	224	13	224	13	224	10	166

Cursor Key Pad

			Shift	Shift	Ctrl	Ctrl	Alt
Key Cap	Scan	ASCII	Scan	ASCII	Scan	ASCII	Scan
Symbol	Code	Code	Code	Code	Code	Code	Code
cp Home	71	224	71	224	119	224	151
cp Up	72	224	72	224	141	224	152
cp PgUp	73	224	73	224	132	224	153
cp Left	75	224	75	224	115	224	155
cp Right	77	224	77	224	116	224	157
cp End	79	224	79	224	117	224	159
cp Down	80	224	80	224	145	224	160
cp PgDn	81	224	81	224	118	224	161
cp Ins	82	224	82	224	146	224	162
cp Del	83	224	83	224	147	224	163

Key Codes and Scan Codes Explanation

(This section explains how key codes and scan codes work.   For a reference chart, see the Key Codes and Scan Codes Table.)

When you press a single key or a key combination, Windows NT translates your keystroke into two numbers: a scan code, representing the actual key that was pressed, and an ASCII code, representing the ASCII value for that key.   Windows NT returns these numbers the next time a program requests keyboard input.   This section explains how key codes work; for information on using them with 4NT see the *4NT.INI* file key mapping directives, keystroke aliases, and INKEY.

Most 4NT commands that use the numeric key codes listed here also use key names, which are usually more convenient to use than the numeric codes.   See Keys and Key Names for more information on key names.

As PCs have evolved, the structure of keyboard codes has evolved somewhat haphazardly with them, resulting in a bewildering array of possible key codes.   We'll give you a basic explanation of how key codes work.   For a more in-depth discussion, refer to a BIOS or PC hardware reference manual.

The nuances of how your keyboard behaves depends on the keyboard manufacturer, the computer manufacturer who provides the built-in BIOS, and your operating system.   As a result, we can't guarantee the accuracy of the information in the tables for every system, but the discussion and reference table should be accurate for most systems.   Our discussion is based on the 101-key "enhanced" keyboard commonly used on 286, 386, 486, and Pentium computers, but virtually all of it is applicable to the 84-key keyboards on older systems.   The primary difference is that older keyboards lack a separate cursor pad and only have 10 function keys.

All keys have a scan code, but not all have an ASCII code.   For example, function keys and cursor keys are not part of the ASCII character set and have no ASCII value, but they do have a scan code.   Some keys have more than one ASCII code.   The **A**, for example, has ASCII code 97 (lower case "a") if you press it by itself.   If you press it along with **Shift**, the ASCII code changes to 65 (upper case "A").   If you press **Ctrl** and **A** the ASCII code changes to 1.   In all these cases, the scan code (30) is unchanged because you are pressing the same physical key.

Things are different if you press **Alt-A**.   **Alt** keystrokes have no ASCII code, so Windows NT returns an ASCII code of 0, along with the **A** key's scan code of 30.   This allows a program to detect all the possible variations of **A**, based on the combination of ASCII code and scan code.

Some keys generate more than one scan code depending on whether **Shift**, **Ctrl**, or **Alt** is pressed.   This allows a program to differentiate between two different keystrokes on the same key, neither of which has a corresponding ASCII value.   For example, **F1** has no ASCII value so it returns an ASCII code of 0, and the **F1** scan code of 59.   **Shift-F1** also returns an ASCII code 0; if it also returned a scan code of 59, a program couldn't distinguish it from **F1**.   The operating system translates scan codes for keys like **Shift-F1** (and **Ctrl-F1** and **Alt-F1**) so that each variation returns a different scan code along with an ASCII code of 0.

On the 101-key keyboard there's one more variation: non-ASCII keys on the cursor keypad (such as up-arrow) return the same scan code as the corresponding key on the numeric keypad, for compatibility reasons.   If they also returned an ASCII code of 0, a program couldn't tell which key was pressed.   Therefore, these cursor pad keys return an ASCII code of 224 rather than 0.   This means that older programs, which only look for an ASCII 0 to indicate a non-ASCII keystroke like up-arrow, may not detect these cursor pad keys properly.

The number of different codes returned by any given key varies from one (for the spacebar) to four, depending on the key, the design of your keyboard, and the operating system.   Some keys, like **Alt**, **Ctrl**, and **Shift** by themselves or in combination with each other, plus **Print Screen**, **SysReq**, **Scroll Lock**, **Pause**, **Break**, **Num Lock**, and **Caps Lock** keys, do not have any code representations at all. The same is true of keystrokes with more than one modifying key, like **Ctrl-Shift-A**. The operating system may perform special actions automatically when you press these keys (for example, it switches into Caps Lock mode when you press **Caps Lock**), but it does not report the keystrokes to whatever program is running. Programs which detect such keystrokes access the keyboard hardware directly, a subject which is beyond the scope of this help file.

Glossary

The glossary contains over 200 terms, and is divided into sections by the first letter of each term.   Select the section you want to review:

4 A B C D E F G H I K L M N O P R S T U V W X

Glossary - 4

4EXIT: A batch file which is executed whenever 4NT exits.

4START: A batch file which is executed whenever 4NT starts.

Glossary - A

Alias Parameter: A numbered variable (*e.g.* %2) included in an alias definition, allowing a different value to be used in the alias each time it is executed.

Alias: A shorthand name for a command or series of commands.

Alternate File Name: See **LFN File System**, and also **SFN**.

AND: A logical combination of two true or false conditions. If both conditions are true, the result is true; if either condition is false, the result is false.

ANSI: Usually a reference to ANSI control sequences, standardized sequences of text characters which control colors on the screen, manipulate the cursor, and redefine keys. 4NT includes support for ANSI screen and cusrsor control sequences. The abbreviation ANSI is for American National Standards Institute, an organization whch sets standards for computer-related systems, including "ANSI" screen control sequences.

Append: Concatenation of one file or string onto the end of another (this use is not related to the DOS, Windows 95 / Windows NT, and OS/2 external command named APPEND).

Application: A program run from the command prompt or a batch file. Used broadly to mean any program other than the command processor; and more narrowly to mean a program with a specific purpose such as a spreadsheet or word processing program, as opposed to a utility.

Archive: A file attribute indicating that the file has been modified since the last backup (most backup programs clear this attribute). Also sometimes refers to a single file (such as a *.ZIP* file) which contains a number of other files in compressed form.

Argument: See **Parameter**.

ASCII File: A file containing ASCII text, as opposed to a binary file which may contain numbers, or other information that cannot be sensibly interpreted as text.

ASCII: The American Standard Code for Information Interchange, which defines numeric values for 128 different characters comprising the English alphabet, numbers, punctuation, and some control characters.

Attribute: A characteristic of a file which can be set or cleared. The standard attributes are Read-Only, Hidden, System, and Archive; other attributes include Directory and Volume Label.

Automatic Batch Files: See **4START** and **4EXIT**.

Automatic Directory Change: A 4NT feature which allows you to change directories by typing the directory name and a backslash [\] at the prompt.

Glossary - B

4 A B C D E F G H I K L M N O P R S T U V W X

Base Name: The file name without a drive, path, or extension. For example, in the file name *C:\DIR1\LETTER.DAT* the base name is *LETTER*.

BAT File: See **Batch File**.

Batch File: A text file containing a sequence of commands for the command processor to execute. Batch files are used to save command sequences so that they can be re-executed at any time, transferred to another system, etc. The extension of a batch file may be *.BAT*, *.CMD*, or *.BTM*, depending on the operating system and command processor you are using.

Batch File Parameter: A numbered variable (*e.g.* %2) used within a batch file, allowing a different value to be used at that spot in the file each time it is executed.

Binary File: A file containing information which does not represent or cannot sensibly be interpreted as text. See also **ASCII File**.

BIOS or **Basic Input Output System**: The software (or "firmware") stored on chips inside PC systems. The BIOS provides basic low-level control of devices required to operate the system, such as the keyboard, floppy disk, and screen; it also handles system self-tests at startup, and intiates loading of the operating system.

Block Device: A physical device for input or output which can transmit or receive large blocks of data while the computer is engaged in other activities. Examples include disk, tape, and CD-ROM drives. See also **Character Device**.

Boot Directory: The current directory at the time the system is booted, usually the root directory of the boot drive.

Boot Drive: The disk drive that the system is booted from, usually A: (the floppy disk) or C: (the hard disk).

Boot: The process of starting the computer and loading the operating system into memory. See also **Reboot**, **Cold Reboot**, and **Warm Reboot**.

Break: A signal sent to a program to tell it to halt what it is doing. The **Ctrl-C** key or **Ctrl-Break** key is used to send this signal. Some external commands abort when they receive a break signal; others return to a previous screen or menu, or abort the current operation.

BTM File: A special type of 4NT batch file which is loaded into memory to speed up execution.

Buffer: An area of memory set aside for storage. For example, disk buffers are used to save information as it is transferred between your program and the disk, and the keyboard buffer holds keystrokes until a program can use them.

Glossary - C

4 A B C D E F G H I K L M N O P R S T U V W X

CDFS or CD-ROM File System: The file system which supports CD-ROM drives. This is typically implemented as a distinct file system in 32-bit operating systems like OS/2 and Windows NT. On other platforms it is implemented as a component of or addition to the underlying general file system for disk drives.

Character Device: A physical device for input or output which must communicate with your computer one character at a time. Examples include the console, communications ports, and printers. See also **Block Device**.

Character Mode: A display mode in which output is displayed in a fixed font, typically with 80 columns in a line and 25 lines on the screen (some systems allow you to increase the number of rows and columns to other fixed sizes), and which cannot display graphics or pictures. See also **Graphics Mode**.

CMD File: See **Batch File**.

CMDLINE: An environment variable used to extend the command line passed to another program beyond its normal length limits.

Cold Reboot: The process of restarting the computer in a way that physically resets most hardware devices, typically by pressing a reset button, or by turning the power off and back on. See also **Warm Reboot.**

Command Completion: A 4NT feature which allows you to recall a previous command by typing the first few letters of the command, then an up-arrow or down-arrow.

Command Echoing: A feature which displays commands as they are executed. Echoing can be turned on and off.

Command Grouping: A 4NT feature which allows you to group several commands with parentheses, and have them treated as a single command for most purposes.

Command History Window: A pop-up window used by 4NT to display the command history, allowing you to choose a previous command to modify and/or execute.

Command History: A 4NT feature which retains commands you have executed, so that they can be modified and re-executed later.

Command Processor: A program which interprets commands and executes other programs. Sometimes also called a **Command Interpreter**.

Command Recall: See **Command History**.

Command Separator: A character used to separate multiple commands on the same command line.

Command Tail: The portion of a command consisting of all the arguments, *i.e.*, everything but the command name itself.

Compound Command: See **Multiple Commands**.

Compression: An operating system feature which compresses data as it is stored in a disk file, and decompresses it as it is read back, resulting in more efficient use of disk space (at a slight cost in processor time to perform the compression and decompression). More generally, an approach to data storage which reduces repeated or redundant information to a smaller number of bytes in the compressed version than in the original, in order to minimize the space required to store the information.

COMSPEC: An environment variable which defines where to find the character-mode command processor to start a secondary shell.

Conditional Commands: A 4NT feature allowing commands to be executed or skipped depending on the results of a previous command. See also **Exit Code**.

Console: The PC keyboard and display.

Console Mode: See **Character Mode**.

Control Character: A character which is part of the ASCII code, but does not have a normal text representation, and which can usually be generated by pressing the Ctrl key along with another key.

Coprocessor: See **Numeric Coprocessor**.

Country Settings: The internal settings which tell the operating system how to interpret keyboard characters which vary from country to country, which character set to use, and how to retrieve and display date, time, and other information in the format appropriate to a particular country. See also **Code Page**.

CPU: The Central Processing Unit which performs all logic and most calculations in a computer. In PC-compatible systems, the CPU is on a single microprocessor chip.

CR or **Carriage Return**: The ASCII character "carriage return" (decimal value 13), generated by pressing the **Enter** key on the keyboard, and stored in most ASCII files at the end of each line.

Critical Error: An error, usually related to a physical or hardware problem with input, output, or network access, which prevents a program from continuing.

Current Directory: The directory in which all file operations will take place unless otherwise specified. The current directory is typically displayed as part of the command prompt. Also called the **Current Working Directory**.

Current Drive: The disk drive on which all file operations will take place unless otherwise specified. The current drive is typically displayed as part of the command prompt.

Cursor: A movable marker on the screen to show where text will be entered when you type at the keyboard, or which object on the screen will be affected when a mouse button is clicked. In character mode only the text cursor is available; graphical systems typically show both a mouse cursor and, when text can be entered, a separate text cursor.

Glossary - D

Date Range: A 4NT feature which allows you to select files based on the date and time they were last modified.

Date Stamp: Information stored in a file's directory entry to show the dates on which the file was created, last modified, and last accessed. Creation and last access dates are not available in the FAT file system. See also **Time Stamp**.

Default Directory: See **Current Directory**

Default Drive: See **Current Drive**.

Delete Tracking: An operating system or utility software feature which is designed to allow you to "undelete" or recover files which have recently been deleted. Delete tracking typically works by temporarily retaining the deleted files and / or information about the deleted files in a special area of the disk.

Description: A string of characters assigned to describe a file with the 4NT DESCRIBE command.

Destination: In file processing commands (*e.g.* COPY or MOVE), the name or directory files should have after any copying or modification has taken place, generally the last specification on the command line. See also **Source**.

Detached Process: A program which is "detached" from the normal means of user input and output, and cannot use the keyboard, mouse, or video display.

Device Driver: A program which allows the operating system to communicate with a device, and which is loaded into memory when the system boots. Device drivers are also used to manage memory or for other similar internal functions.

Device: A physical device for input or output such as the console, a communications port, or a printer. Sometimes "device" is used to refer to character devices, and excludes block devices.

Directive: An individual item in the *4NT.INI* file, used to control the configuration of 4NT.

Directory: A portion of any disk, identified by a name and a relationship to other directories in a "tree" structure, with the tree starting at the root directory. A directory separates files on the disk into logical groups, but does not represent a physical division of the data on the disk.

Directory History: A 4NT feature which allows you to recall recently-used directory names in a popup window, and choose one to switch to.

Directory History Window: See **Directory History**.

Directory Stack: A 4NT feature, implemented through the PUSHD and POPD commands, which allows you to save the current directory and return to it later. See also **Stack**.

Directory Tree: The branching structure of directories on a hard disk, starting at the root directory. The root of the tree is usually considered as the "top" of the structure, so the actual structure can be visualized as an upside-down tree with the root at the top and branches going "down". A portion or branch of the directory tree is sometimes called a "subtree".

DOS Memory: See Base Memory.

DOS Session: See Session.

DPMI or **DOS Protected Mode Interface**: A specification which allows DOS programs to access memory beyond 1 MB in order to manage larger programs or larger amounts of information than will fit in base memory. DPMI support for DOS programs is provided by some DOS memory managers, and by OS/2, Windows 3.1 and above, Windows 95, and Windows NT.

Drive Letter: A letter used to designate a specific local disk volume, or part or all of a network server drive. In most cases drive letters range from A - Z, but some network operating systems allow the use of certain punctuation characters as drive letters in order to support more than 26 volumes.

Glossary - E

Echo: See **Command Echoing**.

Environment: An area of memory which contains multiple entries in the form "NAME=value". See also **Master Environment** and **Passed Environment**.

Environment Variable: The name of a single entry in the environment.

Error Level: A numeric value between 0 and 255 returned from an external command to indicate its result (*e.g.*, success, failure, response to a question). See also **Exit Code**.

Escape Character: In some contexts, the 4NT escape character, which is used to suppress the normal meaning of or give special meaning to the following character. In other cases, the specific ASCII character ESC. The meaning must be determined from the context.

Escape Sequence: A sequence of text characters which has a special meaning and is not treated as normal text. For example, the character sequence <ESC>]K (where <ESC> is the ASCII "escape" character, decimal value 27) will cause an ANSI driver to clear the screen from the cursor to the end of the current line, rather than simply displaying the string "ESC]K" on the screen. Similarly, in 4NT, the escape sequence ^f on the command line is translated to a form feed, and is not treated as the literal characters "^f".

Executable Extensions: A 4NT feature which allows you to specify the application to be executed when a file with a particular extension is named at the command prompt.

Executable File: A file, usually with the extension *.COM* or *.EXE*, which can be loaded into memory and run as a program.

Exit Code: The result code returned by an external command or an internal command. 4NT internal commands return an exit code of 0 if successful, or non-zero if unsuccessful. See also **Errorlevel**.

Expansion: The process 4NT goes through when it scans a command line and substitutes the appropriate actual values for aliases, alias parameters, batch file parameters, and environment variables. See also **Parsing**.

Extended ASCII Character: A character which is not part of the standard set of 128 ASCII characters, but is used on the PC as part of an extended set of 256 characters. These characters include international language symbols, and box and line drawing characters.

Extended Attributes: An OS/2 High Performance File System (HPFS) feature which allows storage of additional information about a file, separate from the file itself. Extended attributes are typically used to store icons for executable files, property or settings information, and other information added by the user.

Extended Directory Search: A 4NT feature which maintains a directory search "database" or list, typically including all directories in your system, and allows you to change quickly to any directory in the list.

Extended Key Code: The code for a key on the PC keyboard which has no representation in the standard ASCII character set, such as a function key, cursor key, or **Alt** plus another key. The extended key code for a key is often the same as the scan code for that key.

Extended Memory: Any memory on a computer system with a 286, 386, 486, or Pentium processor which is above the first 1 MB (one megabyte, or 1024*1024 bytes) of memory. See also **XMS**.

Extended Parent Directory Names: A 4NT feature which allows you to use additional periods in a directory name to represent directories which are successively higher in the directory tree.

Extended Wildcard: A 4NT feature which extends the traditional wildcard syntax and allows you to use multiple wildcard characters, and character ranges (*e.g.* [a-f] for the letters A through F). See also **Wildcard**.

Extension: The final portion of a file name, preceded by a period. For example, in the file name *C:\DIR1\LETTER.DAT* the extension is *.DAT*. In a long filename which contains multiple periods, the extension is usually considered to be the portion of the name after the final period.

External Command: A program which resides in an executable file, as opposed to an internal command which is part of the command processor.

EXTPROC: A command processor feature which allows you to designate a specific external program to run a particular batch file.

Glossary - F

FAT File System: The traditional file system used by DOS to store files on diskettes and hard disks; also supported by OS/2 and Windows NT. Uses a **F**ile **A**llocation **T**able to keep track of allocated and unallocated space on the disk.

FAT-Compatible File Name: See **SFN**.

FF or **Form Feed**: The ASCII character "form feed" (decimal value 12), which typically causes a printer to skip to a new page. The FF character is not normally entered from the keyboard, but in many cases it can be generated, if necessary, by holding the Alt key, pressing 0-1-2, and releasing the Alt key.

File Attribute: See **Attribute**.

File Description: See **Description**.

File Exclusion Range: A 4NT feature which allows you to exclude files from processing by internal commands based on their names.

Filename Completion: A 4NT feature which allows you to type part of a filename on the command line, and have the command processor fill in the rest for you.

Free Memory: Usually, the amount of total memory which is unoccupied and available for applications.

Glossary - G

4 A B C D E F G H I K L M N O P R S T U V W X

Global Aliases: A 4NT option which allows you to store aliases in a global area accessible to all copies of 4NT, so that any change made by one copy is immediately available to all other copies. See also **Local Aliases**.

Global Directory History: An option which allows you to store the directory history in a global area accessible to all copies of 4NT, so that any change made by one copy is immediately available to all other copies. See also **Local Directory History**.

Global History: A 4NT option which allows you to store the command history in a global area accessible to all copies of 4NT, so that any change made by one copy is immediately available to all other copies. See also **Local History**.

Graphics Mode: A display mode in which output is displayed in any one of a range of fonts, typically in resizable windows with a variable number of text rows and columns, and which supports the display of graphics and pictures along with text. See also **Character Mode**.

Glossary - H

Hidden: A file attribute indicating that the file should not be displayed with a normal DIR command, and should not be made available to programs unless they specifically request access to hidden files.

History Window: See **Command History Window** and **Directory History**.

History: See **Command History**.

HMA or **High Memory Area**: The area of PC memory located in the first 64K bytes above the 1 megabyte that DOS can address directly. The HMA can be made addressable from DOS programs using special hardware facilities, or an XMS driver.

HPFS or **High Performance File System**: A file system distributed with OS/2 and Windows NT 3.51 and below which allows longer file names, supports larger drives, and provides better performance than the traditional FAT file system.

Glossary - I

IFS or **Installable File System**: A file system which can be loaded when required to support devices such as CD-ROM or network drives, or non-default disk formats like HPFS (in OS/2) or NTFS (in Windows NT). Installable file systems are primarily supported 32-bit operating systems like OS/2 and Windows NT. Depending on operating system design they may be loaded at boot time, or loaded and unloaded dynamically while the system is running.

Include List: A concise method of specifying several files or groups of files in the same directory, for use with all 4NT commands which take file names as arguments.

Inheritance: A feature which allows one copy of 4NT to "inherit" the *.INI* file data, aliases, command history, and directory history from a previous copy. More generally, a system which allows one program to pass information or settings on to another, often to a second copy of the same program.

.INI File: The 4NT initialization file containing directives which set the initial configuration of the command processor.

Insert Mode: When editing text, a mode in which newly typed characters are inserted into the line at the cursor position, rather than overwriting existing characters on the line. See also **Overstrike Mode**.

Internal Command: A command which is part of the command processor, as opposed to an external command.

Internal Variables: Environment variables created by 4NT to provide information about your system. Internal variables are evaluated each time they are used, and are not actually stored in the environment.

Glossary - K

Key Code: The code passed to a program when a key is pressed on the keyboard. Depending on the key that is pressed, and the software handling the keyboard, the code can be an ASCII code, a scan code, or an extended key code.

Key Mapping: A 4NT feature which allows you to assign new keystrokes for command line functions such as manipulating the command history or completing file names.

Keyboard Buffer: A buffer which holds keystrokes you have typed that have not yet been used by the currently executing program.

Keystroke Alias: An alias assigned to a key, so that it can be invoked or recalled with a single keystroke.

Glossary - L

Label: A marker in a batch file, with the format **:name**, allowing GOTO and GOSUB commands to "jump" to that point in the file. See also **Volume Label**.

LF or **Line Feed**: The ASCII character "line feed" (decimal value 10), stored in most ASCII files at the end of each line, after the CR character. The LF character is not normally entered from the keyboard, but in many cases it can be generated, if necessary, by pressing Ctrl-Enter.

LFN or **Long File Name**: A file name which does not conform to FAT file system restrictions, either because it is longer than the traditional 8 character name plus 3 character extension, or because it contains periods, spaces, or other characters not allowed in a traditional FAT file name. See also **SFN**.

LFN File System: A file system which extends the traditional FAT system to support long filenames and possibly larger hard drives. An LFN file system stores both a long and short name for a file, not just a long name. The short name is sometimes called the "alternate" name. See also **LFN**, **SFN**, **VFAT File System**, and **FAT32 File System**.

Local Aliases: A 4NT option which allows you to store aliases in a local area only accessible to the current copy of 4NT, so that a change made in the current copy of 4NT does not affect other copies, and vice versa. See also **Global Aliases**.

Local Directory History: A 4NT option which allows you to store the directory history in a local area only accessible to the current copy of 4NT, so that a change made in the current copy of 4NT does not affect other copies, and vice versa. See also **Global Directory History**.

Local History: A 4NT option which allows you to store the command history in a local area only accessible to the current copy of 4NT, so that a change made in the current copy of 4NT does not affect other copies, and vice versa. See also **Global History**.

Logging: A 4NT feature, implemented via the LOG command, which allows you to save a record of the commands you execute.

Glossary - M

Master Environment: The master copy of the environment maintained by the command processor.

Modulo: The remainder after an integer division. For example 11 modulo 3 is 2, because when 11 is divided by 3 the remainder is 2.

Multiple Commands: A 4NT feature which allows multiple commands to be placed on a line, separated by a an ampersand [**&**], or another, user-defined character.

Multitasking: A capability of some software (and the related hardware) which allows two or more programs to run apparently simultaneously on the same computer. Multitasking software for PC compatible systems includes operating environments like Windows 3, and complete operating systems like OS/2, Windows 95, and Windows NT.

Glossary - N

4 A B C D E F G H I K L M N O P R S T U V W X

Network: A system which allows several computers to be connected together to share files, printers, modems, or other resources, and to pass electronic mail or other information between the systems on the network.

Network File System: Software which runs over a network to allow access to files on the server. A network file system may support the same options as the file system used on local drives, or it may be more or less restrictive than the local file system about file names, disk volume capacity, and other similar features.

NTFS or **New Technology File System**: A file system distributed with Windows NT which allows longer file names, supports larger drives, and provides better performance than the traditional FAT file system.

Numeric Coprocessor: A chip which works in conjunction with an Intel 8086, 80286, 80386, 80486, or Pentium CPU to perform decimal arithmetic ("floating point") calculations. Some 80486s and the Pentium CPU have the numeric coprocessor built in to the CPU chip; in all other cases it is on a physically separate chip, and is optional (when the coprocessor is not avilable, the CPU performs decimal arithmetic through other, much slower methods).

Glossary - O

Operating System: A collection of software which loads when the computer is started, provides services to other software, and ensures that programs don't interfere with each other while they are running.

Option: See **Switch**.

OR: A logical combination of two true or false conditions. If both conditions are false the result is false; if either condition is true the result is true.

Overstrike Mode: When editing text, a mode in which newly typed characters overwrite existing characters on the line, rather than being inserted into the line at the cursor position. See also **Insert Mode**.

Glossary - P

Parameter: A piece of additional information placed after a command or function name.   For example, in the command DIR XYZ, XYZ is a parameter.   Also used to refer to an alias parameter or batch file parameter.

Parent Directory: The directory in which a particular subdirectory resides, often seen as the directory "above" a subdirectory.

Parsing: The process 4NT performs to analyze the command line, perform alias and environment variable expansion, and find the appropriate internal command or external command to execute.   More generally, the process of breaking down a string or message into its individual components in order to process them properly.

Passed Environment: A copy of the master environment created before running an application, so that any changes made by the application will not affect the master environment.

Path: A specification of all the directories a file resides in.   For example, the path for *C:\WPFILES\MYDIR\MEMO.TXT* is *C:\WPFILES\MYDIR\*.   Also used to refer to the environment variable PATH, which contains a series of path specifications used when searching for external commands and batch files.

Pipe: A method for collecting the standard output of one program and passing it on as the standard input of the next program to be executed, signified by a vertical bar "|" on the command line.   See also **Redirection**.

Previous Working Directory: The working directory used most recently, just prior to the current working directory.   For example, if *C:\DATA* is the current working directory and you switch to *D:\UTIL*, *C:\DATA* then becomes the previous working directory.

Primary Shell: The copy of the character-mode command processor which is loaded by the operating system when the system boots or a session opens.

Glossary - R

RAM or **Random Access Memory**: The physical memory used to store data while a computer is operating. The information in most types of RAM is lost when power is turned off.

RAM Disk: A pseudo "disk drive", created by software, which appears like a normal physical disk drive to programs. Sometimes also called a **Virtual Disk**.

Range: See **Date Range**, **Size Range**, **Time Range**, and **File Exclusion Range**.

Read-Only: A file attribute indicating that the file can be read, but not written or deleted by the operating system or the command processor unless special commands are used.

Reboot: The process of restarting the computer with software, with the keyboard (*e.g.* by pressing Ctrl-Alt-Del), by pressing a reset button, or by turning the power off and back on. See also **Cold Reboot** and **Warm Reboot**.

Redirection: A method for collecting output from a program in a file, and/or of providing the input for a program from a file. See also **Pipe**.

Registry: A hierarchically organized data file maintained by Windows 3.x, Windows 95, and Windows NT to hold system parameters, hardware and software settings, and other similar information used by the operating system or by other software packages.

REXX: A file and text processing language developed by IBM, and available on many PC and other platforms.

ROM or **Read Only Memory**: A physical memory device used to store information which cannot be readily modified, such as the BIOS built into each PC system. The information in ROM is typically retained when power is turned off.

Root Directory: The first directory on any disk, from which all other directories are "descended." The root directory is referenced with a single backslash [\].

Glossary - S

Scan Code: The physical code for a key on the PC keyboard. For the original U.S. English keyboard layout the scan code represents the physical position of the key, starting with 1 for the key in the upper left corner (Esc), and increasing from left to right and top to bottom. This order will vary for more recent keyboards or those designed for other countries or languages.

Search Path: See **PATH**.

Secondary Shell: A copy of the command processor which is started by another program, rather than by the operating system.

Session: A general term for the individual windows or tasks started by a multitasking system. For example, under Windows NT you might run a Windows NT application in one session, and 4NT in another.

SFN or **Short File Name**: A file name which follows the rules of the traditional FAT file system: a name of 1 - 8 characters and an extension of 0 - 3 characters, each consisting of only alphabetic and numeric characters plus the punctuation marks **! # $ % & ' ( ) - @ ^ _ ` { }** and **~**. See also **LFN**.

Shell: See **Command Processor**. Also used to refer to a program which gives access to operating system functions and commands through a menu- or mouse-driven system, or which replaces the primary user interface of the operating system.

Size Range: A 4NT feature which allows you to select files based on their size.

Source: In file processing commands (*e.g.* COPY or MOVE), the original files before any copying or modification has taken place, *i.e.*, those specified earlier on the command line. See also **Destination**.

Stack: An area of memory used by any program to store temporary data while the program is running; more generally, any such storage area where the last item stored is normally the first one removed.

Standard Error, Standard Input, and **Standard Output**: The file(s) or character device(s) where a program respectively displays error messages, obtains its normal input, and displays its normal output. Standard error, standard input, and standard output normally refer to the console, unless redirection is used.

Subdirectory: Any directory other than the root directory.

Subtree: See **Directory Tree**.

Swap File: A disk file created by an operating system or a program to store unused information on disk, and thereby free up memory for other purposes.

Switch: A parameter for an internal command or application which specifies a particular behavior or setting. For example, the command "DIR /P" might be referred to as "having the /P switch set".

System: A file attribute indicating that the file belongs to the operating system or command processor, and should not be accessed by other programs.

Glossary - T

Target: See **Destination**.

Time Range: A 4NT feature which allows you to select files based on the time they were last modified.

Time Stamp: Information stored in a file's directory entry to show the times at which the file was created, last modified, and last accessed. Creation time is not available in the FAT file system; last access time is only available in the HPFS and NTFS file systems. See also **Date Stamp**.

Tree: See **Directory Tree**.

Glossary - U

UMB or **Upper Memory Block**: An XMS Upper Memory Block, whose address is above the end of base memory (normally, above 640K), but within the 1 megabyte of memory that DOS can address directly.

UNC or **Universal Naming Convention**: A common method for accessing files on a network drive without using a "mapped" drive letter. Names specified this way are called UNC names, and typically appear as **\\server\volume\path\filename**, where **server** is the name of the network server where the files reside, **volume** is the name of a disk volume on that server, and the **path\filename** portion is a directory name and file name.

Glossary - V

4 A B C D E F G H I K L M N O P R S T U V W X

Variable Expansion: The process of scanning a command line and replacing each environment variable name, alias parameter, or batch file parameter with its value.

Variable Functions: Functions provided by 4NT to manipulate strings, dates, and filenames; perform arithmetic; read and write files; and perform other similar functions. Variable functions are similar to static environment variables or internal variables, but have parameters and can perform actions rather than just returning static information.

Variable: See **Alias Parameter**, **Batch File Parameter**, and **Environment Variable**.

VFAT File System: An extension of the FAT file system, available in Windows 95 and Windows NT, which supports long filenames. Also see **LFN** and **FAT32 File System**.

Virtual Disk: See **RAM Disk**.

Volume Label: A special, hidden file placed on any disk, whose name constitutes a "label" for the entire disk.

Volume: See **Disk Drive**.

Glossary - W

4 A B C D E F G H I K L M N O P R S T U V W X

Warm Reboot: The process of restarting the computer with software, or with the keyboard (*e.g.* by pressing Ctrl-Alt-Del), typically without physically resetting any hardware devices. See also **Cold Reboot**.

White Space Character: A character used to separate arguments on the command line. The white space characters recognized by 4NT are the space, tab, and comma.

Wildcard: A character ("*" or "?") used in a filename to specify the possibility that any single character ("?") or sequence of characters ("*") can occur at that point in the actual name. See also **Extended Wildcard**.

Windows NT File System: See **NTFS**.

Glossary - X

XOR (exclusive OR): A logical combination of two true or false conditions. If both conditions are false or both conditions are true the result is false; if either condition is true and the other is false the result is true.