Table of Contents

About 4DOS/NT Help

4DOS for Windows NT

Version 2.5 Help System

Text by Hardin Brothers, Tom Rawson, and Rex Conn

[9/94-2.5A]

Using the 4DOS/NT Help System

This online help system for 4DOS/NT covers all 4DOS/NT features and internal commands. It includes reference information to assist you in using 4DOS/NT and developing batch files; however it does not include all of the details which are included in the printed 4DOS/NT manuals.

If you type part or all of a command on the line and then press **F1**, 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 the list of all help topics and you can pick the one you want.

You can use this feature to obtain help on any topic -- not just on commands. For example, if you enter the command **HELP _DISK** you will see help for the **_DISK** internal variable.

If you type the name of any internal command at the prompt, followed by a slash and a question mark [**/?**] like this:

```
copy /?
```

then you will also see help for the command.

The **/?** option may not work correctly if you have used an alias to redefine how an internal command operates. To view the **/?** help for such a command you must add an asterisk to the beginning of the command to disable alias processing. For example, if you have defined this alias:

```
alias copy *copy /r
```

then the command COPY /? will be translated to COPY /R /?, which will not work properly. However, if you use *COPY /?, the alias will be ignored and the **/?** will work as you intended.

4DOS/NT 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.

Conventions

This section contains information about conventions that are used throughout 4DOS/NT:

Colors and color names

Color-coded directories

Keys and key names

These topics are combined here in a central reference spot so that they will be easy to find when you need to refer to them. You will find cross references to this section throughout the help system.

Colors and Color Names

You can use color names in several of the directives in the .INI file 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. 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 4DOS/NT color specifications.

Color-Coded Directories

The DIR and SELECT commands can display each file name in a different color, depending on the file's extension.

To choose the DIR and SELECT display colors, you must either use the SET command to create an environment variable called COLORDIR, or use 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 and SELECT will use the default screen colors or those set with the SelectColors directive.

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
RDONLY	Read-only files
HIDDEN	Hidden files
SYSTEM	System files
ARCHIVE	Files modified since the last backup

and "ColorName" is any valid color name (see Colors).

Unlike most color specifications, the background portion of the color name may be left out for directory colors. If you don't specify a background color, DIR and SELECT 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 bright green on white (this should be entered on one line):

```
[c:\] set colordir=com exe:red;c asm:bri cyan;rdonly:bri gre on whi
```

Extended wildcards (for example "BA[KXC]" for .BAK, .BAX, and .BAC files) can be used in directory color specifications.

If the extension or type of a file matches an extension or type listed in your color specification, the remainder of the colors are ignored for that file. You may need to take this into account in determining the order of extensions and file types in your COLORDIR setting. For example, you might try this setting to display all *.COM* and *.EXE* files in red, and all other files whose extension starts with a "C" in green:

```
c:\> set colordir=c*:green;com exe:red
```

However in this case the *.COM* files will be displayed in green, because they match the "c*", and the ".com" later on the line is ignored. To correct this problem, change the line to read:

```
c:\> set colordir=com exe:red;c*:green
```

Keys and Key Names

Key names are used to define keystroke aliases, and in several 4NT.INI 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-case or lower-case. You cannot specify a punctuation key.

The prefix and key name must be separated by a dash [**-**]. For example:

Alt-F10This is okay
Alt F10The 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 4DOS/NT. 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 4DOS/NT 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 and not to any problem with 4DOS/NT.

Startup

You will typically start 4DOS/NT from an object on your Windows NT desktop. You can create as many 4DOS/NT objects as you wish on the desktop. Different objects can be used to start 4DOS/NT 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 4DOS/NT window. You can set any necessary command line parameters for 4DOS/NT 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 4DOS/NT is included later in this section.

For general information on creating and configuring desktop objects, see your Windows NT documentation.

When you configure a 4DOS/NT 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 4DOS/NT (*e.g.*, **@inifile**) after the *4NT.EXE* file name. For example:

Command Line:	D:\4NT20\4NT.EXE @D:\4NT.INI
Working directory:	C:\

To run a startup batch file for a particular 4DOS/NT 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 4DOS/NT commands. You can also execute any internal 4DOS/NT 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:\4NT20\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:\4NT20\4NT.EXE /C COMFILES.BTM
Working directory:	C:\

The 4DOS/NT command line does not need to contain any information. When invoked with an empty command line, 4DOS/NT 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 4DOS/NT command line that will affect the way it operates.

Command line options for primary shells are set in the Command Line field of the 4DOS/NT object. Command line options for secondary shells can be set on the secondary shell command line.

4DOS/NT 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] [/C | /K] [command]

The options are:

d:\path: 4DOS/NT will use this directory and path to set the COMSPEC environment variable for this session. If this option is not used, COMSPEC is set from the location of 4NT.EXE. 4DOS/NT always knows what drive and directory it was started from and can set COMSPEC accordingly. This option is included only for compatibility with CMD.EXE. This option cannot be used for secondary shells.

@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 is stored in the same subdirectory as 4NT.EXE or in the root directory of the boot drive.

//iniline: This option tells 4DOS/NT to treat the text appearing between the **//** and the next space or tab as a 4NT.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. This option overrides any corresponding directive in your 4NT.INI file.

/L, **/LA**, **/LD**, and **/LH:** These options force 4DOS/NT to to use a local alias, directory history, and / or command history list. They can be used to override any LocalAlias=No, LocalDirHistory=No, or LocalHistory=No settings in *4NT.INI*. This allows you to use global lists as the default, but start a specific 4DOS/NT session with local lists. 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 history, and **/L** forces all three.

/Q: This option has no effect. It is included only for compatibility with *CMD.EXE*.

/S: This option tells 4DOS/NT 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.

[/C | /K] command: This option tells 4DOS/NT to run a command when it starts. The command will be run after 4START has been executed and before any command prompt is displayed. It can be any valid internal or external command, batch file, or alias; you may include multiple commands by using the command separator. All other startup options must be placed before the command, because 4DOS/NT 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**, 4DOS/NT will execute the command and then exit and return to the parent program or the Windows NT desktop without displaying a prompt.

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.

To run a startup batch file for a particular 4DOS/NT session, include its name (with a path, if the batch file is not in the session's startup directory) as the last item in the Command Line field when you configure the desktop object. 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 4DOS/NT commands.

The Command Line

4DOS/NT 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 4DOS/NT. 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:

Additional command-line features are documented under Redirection and Piping and File Selection.

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 extends to a maximum of 1023 characters.

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

4DOS/NT 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 above 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 4DOS/NT 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.

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

Most of the command-line editing capabilities are also available when a 4DOS/NT 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.

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.
@	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 "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 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 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.

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 to the end of the list when it is re- executed, set HistCopy to Yes in your *.INI* file.

Local and Global Command History

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

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

With a global history list, all copies of 4DOS/NT 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 4DOS/NT.

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

If you select a global history list for 4DOS/NT you can share the history among all copies of 4DOS/NT running in any session. When you close all 4DOS/NT sessions, the memory for the global history list is released, and a new, empty history list is created the next time you start 4DOS/NT. If you want the history list to be retained in memory even when no command processor session is running, you need to load the SHRALIAS program, which performs this service for the global history and alias lists. SHRALIAS is described in more detail under the ALIAS command.

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.

You can control the position and size of the history window with configuration directives in *4NT.INI*. You can also change the keys used in the window with key mapping directives in the *.INI* file.

Filename Completion

Filename Completion Keys:

F8 Get the previous matching filename.
or **Shift-Tab**

F9 Get the next matching filename.
or **Tab**

F10 Keep the current matching filename and display the next matching name immediately after the current one.

Filename completion can help you by filling in a complete file name on the command line when you only remember 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. 4DOS/NT 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 4DOS/NT 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**) and **Shift-Tab** (or **F8**) to move through the remaining matching files.

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 matching pattern will be *.*. If you type a filename without an extension, 4DOS/NT will add *.* to the name. 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.

Filename Completion Window

You can also view filenames in a scrollable **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 the names 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. (**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.

Directory History Window

Directory History Window Keys:

Ctrl-PgUp (from the command line) Open the directory history window.
or **Ctrl-PgDn**

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 Go to the beginning of the directory list.
or **Home**

Ctrl-PgDn Go to the end of the directory list.
or **End**

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.

Every time you change to a new directory or drive, the current directory is recorded in an internal directory history list. You can set the size of the list with the DirHistory directive in the *.INI* file. As new entries are added, old entries are deleted from the list. Directory changes are recorded whether you make them from the command line with the CD, CDD, PUSHD, or POPD commands, with an automatic directory change, or by typing a new drive letter followed by a colon. Directories are recorded whether you change from one to another at the command line, from within a batch file, or from within an alias. 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.

You can view the directory history from the scrollable **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.

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 4DOS/NT.

You can control the type of directory history list with the LocalDirHistory directive in the *.INI* file, 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 (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 4DOS/NT running in any session. When you close all 4DOS/NT sessions, the memory for the global directory history list is released, and a new, empty list is created the next time you start 4DOS/NT. If you want the list to be retained in memory even when no command processor session is running, you need to load the SHRALIAS program, which performs this service for the global command history, directory history, and alias lists. SHRALIAS is described in more detail under the ALIAS command.

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.

Automatic Directory Changes

The automatic directory change feature gives you a quick method for changing directories. You can use an automatic directory change in place of the 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 feature can make directory changes very simple when it's combined with CDPATH, a list of directories for the CD and CDD commands to search if the directory you name does not exist below the current directory. For example, suppose CDPATH is set to C:\;D:\;E:\, 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]
```

In executing the command shown above, 4DOS/NT first looks for a WIN subdirectory of the current directory, *i.e.*, C:\4NT\WIN. If no such directory exists it looks for a WIN subdirectory in every directory in the CDPATH list, and changes to the first one it finds.

Internally, automatic directory changes use the CDD command, so the text before the backslash can include a drive letter, a full path, or a partial path. 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.

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 batch files and alias definitions 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 4DOS, 4OS2, and 4DOS/NT, see 4DOS, 4OS2, and 4DOS/NT Compatibility for details about choosing compatible command separators for two or more products.

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 4DOS/NT 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 an error message and the remainder of the line will not be executed.

Page and File Prompts

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

When 4DOS/NT 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 "Yes, process this file;" **N** for "No, do not process this file;" or **R** or **Esc** for "process the Remainder of the files without further prompting." You can also press **Ctrl-C** or **Ctrl-Break** at this prompt to cancel the remainder of the command.

Conditional Commands

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.

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 copy then rename all the *.WKQ* files on drives A: and B: using the FOR command. You could do it like this:

```
[c:\] for %drv in (A B) do copy %drv:*.wkq d:\wksave\
[c:\] for %drv in (A B) do ren %drv:*.wkq *.old
```

But with command grouping you can do the same thing in one command (enter this on one line):

```
[c:\] for %drv in (A B) do (copy %drv:*.wkq d:\wksave\ & ren %drv:*.wkq *.sav)
```

The COPY and REN commands enclosed in the parentheses appear to FOR as if they were a single command, so both commands are executed for every element of the FOR list.

You can also use command grouping to redirect input or output for several commands without repeatedly using the redirection symbols. For example, consider the following batch file fragment which uses the ECHO command to create a file (with **>**), and to append to the file (with **>>**):

```
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.

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. 4DOS/NT displays a "More?" prompt after each incomplete line:

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

Escape Character

4DOS/NT 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 or the EscapeChar directive in your *.INI* file. If you plan to share aliases or batch files between 4DOS, 4OS2, and 4DOS/NT, see 4DOS, 4OS2, and 4DOS/NT Compatibility for details about choosing compatible escape characters for two or more products.

Eight 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 useful for redirecting codes to the printer, and **^r** can be used in keystroke aliases. The special characters which can follow the escape character are:

- **b** backspace
- **c** comma
- **e** the ASCII ESC character (ASCII 27)
- **f** form feed
- **n** line feed
- **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 send a form feed followed by the sequence ESC Y to the printer, you can use this command:

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

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, 4DOS/NT, or your application program cannot proceed until the error is resolved.

When a critical error occurs, you will see a popup window asking you to choose an error handling option. The message comes from Windows NT, and will typically offer you three choices:

Abort Tell the program that the operation failed. This option returns an error code to 4DOS/NT or to the application program that was running when the error occurred. 4DOS/NT generally stops the current command when an operation fails.

Retry Choose this option if you have corrected the problem.

Ignore Ignore the error and continue. This option can be dangerous; it tells 4DOS/NT (or the application that was running when the error occurred) that the operation succeeded when it didn't!

Redirection and Piping

This section covers **redirection** and **piping**. You can use these features to change how 4DOS/NT 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

Redirection assigns **standard input**, **standard output**, and **standard error** to a device like the printer or serial port, or to a file.

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.

Here are the standard redirection options supported by 4DOS/NT (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

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:

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

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.

If you want to append output to the end of an existing file, rather than creating a new file, replace the first "**>**" in the output redirection symbol with "**>>**" (use **>>**, **>>&**, and **>>&>**).

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 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. Setting NoClobber will also prevent the creation of a new file.

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, 4DOS/NT 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 have opened a file with the %@FILEOPEN variable function and the file handle is between 3 and 9.

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 /s
```

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.

4DOS/NT 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 is ended automatically.

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

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.

The features explained in this section apply to 4DOS/NT 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

4DOS/NT 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.

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 4DOS/NT, 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 are not likely to work properly with software other than 4DOS/NT.

"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 /l"[]" *.btm
```

which will display a list of all .BTM files which **dont** have a description.

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 NTFS filenames containing semicolons [**;**], and the use of a semicolon to indicate an include list.

For example, if you have a file on an NTFS drive 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, 4DOS/NT 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 *C:\DATA\LETTER1;V2*:

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

Date, Time, and Size Ranges

Most internal commands which accept wild cards also allow date, time, and size ranges to further define the files that you wish to work with. 4DOS/NT will examine the files' time stamps (which record when the file was last modified), and the files' sizes, to determine which files meet the range criteria that you specify.

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 ("**]**").

See the individual range types for details on specifying ranges:

Size Ranges

Date Ranges

Time Ranges

Using Ranges

If you combine two types of ranges, a file must satisfy both ranges to be included. For example, **/[d2-8-94,2-9-94] /[s1024,2048]** means files last modified between February 8 and February 9, 1994, 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, 1994, you could use this command:

```
[c:\] dir /[d10-1-94,+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:
```

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

The HPFS and NTFS file systems maintain 3 sets of dates and times for each file: creation, last access, and last modification. By default, date time ranges work with the last modification time stamp. You can use the "last access" (a) or "created" (c) time stamp in a date 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]

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

Date Ranges

Date ranges select files that were created or last modified at any time between the two dates. For example, **/[d12-1-94,12-5-94]** selects files that were last modified between December 1, 1994, and December 5, 1994.

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 **/[d7-1-95@8:00a,7-3-95@6:00p]** selects files that were modified at any time between 8:00 am on July 1, 1995 and 6:00 PM on July 3, 1995. 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, 4DOS/NT substitutes the current date and time. For example, **/[d10-1-94]** selects files dated between October 1, 1994 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
/[d10-27-94,+3]	modified between 10-27-94 and 10-30-94
/[d10-27-94,-3]	modified between 10-24-94 and 10-27-94
/[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)

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, **/[d12-8-94@12:00,+2@12:00]** selects files that were last modified between noon on December 8 and noon on December 10, 1994. 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.

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

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 match 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 filename feature 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 the SELECT command for another way 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:
```

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 it).

Executable Extensions

The syntax for creating an executable extension is:

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

This tells 4DOS/NT 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, 4DOS/NT 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 this default list of extensions, and have 4DOS/NT 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.

Environment variables 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, 4DOS/NT 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 *QBASIC.EXE* as the processor for *.BAS* files:

```
[c:\] set .bas=c:\dos\qbasic.exe /run
```

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

```
[c:\] pushcart
```

4DOS/NT will execute the command:

```
c:\dos\qbasic.exe /run pushcart.bas
```

The next example defines *B.EXE* (the Brief text editor) as the processor for *.C* files:

```
[c:\] set .c=c:\brief\b.exe -Mxyz
```

Now, if you have a file called *HELLO.C* and enter the command

```
[c:\] hello -i30
```

This will be expanded to:

```
c:\brief\b.exe -Mxyz hello.c -i30
```

Notice that the text from the .C environment variable is inserted at the beginning of the line, including any options, followed by the original file name plus its extension, and then the remainder of the original command line.

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.

Batch Files

A batch file is a file that contains a list of commands to execute. 4DOS/NT 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:

.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), and for batch files larger than 64K bytes.

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.

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 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
```

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

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

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 4DOS and 4DOS/NT, you can select a new character for any product with the SETDOS /P command or the ParameterChar directive in your *.INI* file.

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
```

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

Automatic Batch Files

Each time 4DOS/NT 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 4DOS/NT itself, you should use the 4StartPath directive in your *.INI* file to specify its location. *4START* is optional, so 4DOS/NT will not display an error message if it cannot find the file.

Whenever a 4DOS/NT shell ends, it runs a third 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.

Detecting 4DOS/NT

From a batch file, you can determine if 4DOS/NT is loaded by testing for the variable function @EVAL, with a test like this:

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

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

Batch File Compression

You can compress your .BTM files with a program called *BATCOMP.EXE*, which is distributed with 4DOS/NT. 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.

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.

Each of our command processors includes its own version of *BATCOMP.EXE*, set up to run under the corresponding operating system. However, the output produced by each program is the same, so a batch file compressed with any version of BATCOMP can be used with any JP Software command processor.

If you plan to distribute batch files to users of different platforms, see 4DOS, 4OS2, and 4DOS/NT Compatibility.

Argument Quoting

As it parses the command line, 4DOS/NT 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 4DOS/NT 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.

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.

4DOS, 4OS2, and 4DOS/NT 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 (in this section, **<Ctrl-X>** stands for the ASCII Ctrl-X character, numeric value 24. This character appears on your screen as an up-arrow [].):

Character	4DOS Default	4DOS/NT and 4OS2 Default
Command Separator	^	&
Escape Character	<Ctrl-X>	^
Parameter Character	&	$

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

1) Select a consistent set of characters with *.INI* file configuration directives or the SETDOS command. For example, to set the 4DOS/NT characters to match 4DOS, use these lines in *4NT.INI*:

```
CommandSep = ^
EscapeChar = <Ctrl-X>
ParameterChar = &
```

2) 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)
```

The following chart shows the correspondence between the appropriate SETDOS command options, *.INI* file directives, and internal variables:

Special Character	SETDOS Switch	INI File Directive	Internal Variable
Command Separator	/C	CommandSep	%+
Escape Character	/E	EscapeChar	%=
Parameter Character	/P	ParameterChar	(none)

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 4DOS/NT batch language, especially if you need advanced string processing capabilities.

The REXX language is not built into 4DOS/NT. You can use Personal REXX for Windows NT, developed by Quercus Systems of Saratoga, CA. (Personal REXX is available from JP Software or directly from Quercus Systems.)

REXX programs are stored in *.CMD* files. When 4DOS/NT loads, it searches for the Personal REXX .DLL's and loads them if found. 4DOS/NT checks to see if the first two characters on the first line of a *.CMD* file are [**/***], the beginning of a REXX comment. If so, it passes the file to Personal REXX for processing.

EXTPROC Support

4DOS/NT 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. 4DOS/NT 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
```

4DOS/NT 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 4DOS/NT 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.

The Environment

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 4DOS/NT:

CDPATH

CMDLINE

COLORDIR

COMSPEC

PATH

PROMPT

4DOS/NT also supports two special types of variables. Internal variables are similar to environment variables, but are stored internally within 4DOS/NT, 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
```

The variable names you use this way 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, 4DOS/NT uses the environment to keep track of the default directory on each drive. DOS and OS/2 keep track of the default directory for each drive letter internally; Windows NT does not. 4DOS/NT 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 can view these variables with the SET command, but you cannot change them.

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

The trailing percent sign that was traditionally required for environment variable names is not usually required in 4DOS/NT, which accept any character that cannot be part of a variable name 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.

CDPATH

CDPATH tells 4DOS/NT where to search for directories specified by the CD, CDD, and PUSHD commands and in automatic directory changes. (**_CDPATH** can be used as an alternative to CDPATH if you are using Microsoft Bookshelf, which uses a CDPATH variable for its own purposes.)

CDPATH is composed of a list of directories, separated by semicolons [**;**]. If CD, CDD, PUSHD, or an automatic directory change can't locate the specified directory to change to, they will append the specified directory name to each directory in CDPATH and attempt to change to that drive and directory, until the first match or the end of the CDPATH argument. This allows you to use CDPATH as a quick way to find commonly used subdirectories which have unique names. For example, if you are currently in the directory C:\WP\LETTERS\JANUARY and you'd like to change to D:\SOFTWARE\UTIL, you could enter the command:

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

However, if the D:\SOFTWARE directory is listed in your CDPATH variable, and is the first directory in the list with a UTIL subdirectory, you can simply enter the command

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

to change to D:\SOFTWARE\UTIL.

You can create CDPATH with the SET command. 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:\
```

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 4DOS/NT. For example, if 4DOS/NT is stored in the directory C:\4DOS/NT, the COMSPEC variable should be set to C:\4NT\4NT.EXE. COMSPEC is used by applications which need to find 4DOS/NT to implement a "shell to the command prompt" feature.

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

PATH

PATH is a list of directories that 4DOS/NT 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 4DOS/NT 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 4DOS/NT, 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.

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

Command processor status

_4VER	4DOS/NT 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	4DOS/NT 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 and color

_BG	Background color at cursor position
_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.)
_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
_SYSERR	* Last Windows NT error code

Compatibility

=	* Substitutes escape character
+	* Substitutes command separator

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 4DOS/NT) 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 4DOS/NT) 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 4DOS/NT version (for example, "2.5").

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

_BATCH is the current batch nesting level. It 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 pathname 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.

_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

_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 day of the month (1 to 31).

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

_DOS is the operating system type ("DOS", "OS2", or "NT"). 4DOS always returns "DOS", 4OS2 always returns "OS2", and 4DOS/NT always returns "NT". This may be useful if you have batch files running under more than one operating system.

_DOSVER is the current operating system version (for example, "3.5").

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

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

_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).

_HOUR is the current hour (0 - 23).

_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).

_MINUTE is the current minute (0 - 59).

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

_MOUSE always returns "1" in 4DOS/NT.

_NDP is the coprocessor type:

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

_PID is the current process ID number.

_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 Startup Options 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.

_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]	Closes 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

@EXT[filename]	File extension
@FILENAME[filename]	File name and extension
@FULL[filename]	Full file name with path
@NAME[filename]	File name without path or extension
@PATH[filename]	File path without name

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
@INSTR[start,length,string]	Extract a substring
@LEN[string]	Length of a string
@LOWER[string]	Convert string to lower case
@REPEAT[c,n]	Repeat a character
@SUBSTR[string,start,length]	Extract a substring
@TRIM[string]	Remove blanks from a string
@UPPER[string]	Convert string to upper case
@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
@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
@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
@TIME[hh:mm:ss]	Convert time to number of seconds

Utility

@ALIAS[name]	Value of an alias
@EXEC[command]	Execute a command
@IF[condition,true,false]	Evaluates a test condition
@REXX[expr]	Execute a REXX expression
@SELECT[file,t,l,b,r,title]	Menu selection
@TIMER[n]	Elapsed time of specified timer

Like all environment variables, these variable functions must be preceded by a percent sign (%@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.

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 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.

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, @FILESEEK, 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 4DOS/NT (or, in a secondary shell, the program which started 4DOS/NT), or hang your system.

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.

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[%&]`
```

@ALTNAME[filename]: Returns the alternate (8.3 FAT-format) name for the specified file. Under Windows NT 3.1 this function returns a valid name on HPFS and NTFS drives only. Under Windows NT 3.5 and above it works on all drives.

@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.

@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]]: 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	Directory
A	Archive

The attributes (other than **N**) can be combined (for example **%@ATTRIB[*MYFILE*,HS]**). ATTRIB will only return a 1 if all of the attributes 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.

@COMMA[n]: Inserts commas, or the "thousands separator" character for your country ID, into a numeric string.

@DATE[mm-dd-yy]: Returns the number of days since January 1, 1980 for the specified date. DATE uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan).

@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 size of the largest free memory block (either in physical or virtual memory).

@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.

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 with the EvalMax and EvalMin *4NT.INI* directives and with the SETDOS /F command.

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. 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, .BTM file, or .BAT 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.

@EXT[filename]: Returns the extension from a file name, without a leading period.

@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.

@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]: Returns the date a file was last modified, in the default country format (mm-dd-yy for the US).

@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)	S	System
R	Read-only	D	Directory
H	Hidden	A	Archive

The attributes (other than **N**) can be combined (for example **%@FILES[*MYFILE*,HS]**). @FILES 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."

@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". 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.

@FILESIZE[filename,b|k|m]: 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.

@FILETIME[filename]: Returns the time a file was last modified, in hh:mm format.

@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.

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** Directory
- **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. @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 (see @FINDFIRST for details). Returns an empty string when no more files match. @FINDNEXT should only be used after a successful call to @FINDFIRST.

@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.

@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, "NTFS" for a drive that uses Windows NT's high performance 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 name of a file.

@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, **%IF[2==2,Correct!,Oops!]** returns "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.

@INSTR[start, length, string]: The same as @SUBSTR. However, the string is at the end of the @INSTR argument list, so that commas in the string will not be confused with commas separating the arguments.

@INT[n]: Returns the integer part of the number n.

@LABEL[d:]: Returns the volume label of the specified disk drive.

@LEN[string]: Returns the length of a string.

@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. If you need to scan through the lines of a file in sequence, the @FILEREAD function (above) and the "@filename" construct available in the FOR command are much faster than calling the @LINE function repeatedly. @LINE will 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. 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.

@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!
```

@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.

@NAME[filename]: Returns the base name of a file, without the path or extension.

@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".

@PATH[filename]: Returns the path from a file name, including the drive letter and a trailing backslash but not including the base name or extension.

@RANDOM[min, max]: Returns a random value between min and max, inclusive. Min, max, and the returned value are all integers.

@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.

@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 (*i.e.*, a floppy disk or removable hard disk); otherwise returns "0".

@REPEAT[c,n]: Returns the character "c" repeated "n" times.

@REXX[expr]: Calls the REXX interpreter to execute the expression. Returns the result string from REXX; if the REXX expression does not return a string, **@REXX** returns the REXX numeric result code.

@SEARCH[filename]: Searches for the filename using the PATH environment variable, 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]: Pops up a selection window with the lines from the specified file. Returns the text of the line the scrollbar is on if you press **Enter**, or an empty string if you press **Esc**. @SELECT can be used to display menus or other selection lists from a batch file. To select from lines passed through input redirection or a pipe, use CON as the filename. You can move through the selection window with standard navigation keystrokes. To change the navigation keys, see the Key Mapping directives in the *.INI* file.

@SUBSTR[string,start,length]: 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, %@SUBSTR[%_TIME,0,2] gets the current time and extracts the hour. 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. @INSTR performs the same function, and allows commas in the string without quoting.

@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.

@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 whether the file is created on a FAT, NTFS, or HPFS drive. This function allows you to create a temporary file without overwriting an existing file.

@UPPER[string]: Returns the string converted to upper case.

@WORD[["xxx",]n,string]: Returns the "nth" 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 separators that you wish to use. If you want to use a double quote as a separator, prefix it with an escape character (see page 71). If you dont specify a list of separators, @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 separators.

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. The optional list of delimiters follows the same format as @WORD. If the string argument is enclosed in quotation marks, you **must** enter a list of delimiters as well.

4NT.INI

The configuration of 4DOS/NT is controlled through an optional file of initialization information called *4NT.INI*.

This section contains general information on 4NT.INI, and an example. 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.

You can create, add to, and edit the .INI file with any ASCII text editor. Each command processor reads its .INI file when it starts, and configures itself accordingly. The .INI file is not re-read when you change it. For changes to take effect, you must restart the session or window in which 4DOS/NT is running.

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. If you are happy with all of the default values, you don't need an .INI file at all.

4DOS/NT primary shells search for the *.INI* file in three places:

1) If there is an "@d:\path\inifile" option on the 4DOS/NT startup command line 4DOS/NT will use the path and file name specified there, and will not look elsewhere.

2) If there is no *.INI* file name on the startup command line, the search proceeds to the same directory where the 4DOS/NT program file (*4NT.EXE*) is stored. This is the "normal" location for the *.INI* file. 4DOS/NT determines this directory automatically. You can also set it yourself by placing a COMSPEC directory name on the startup command line.

3) 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 4DOS/NT 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, and then processes any "@d:\path\inifile" option on the secondary shell command line. You can override this behavior with the NextINIFile directive.

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 "History" is the directive and "2048" is the value:

```
History = 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:

```
History = 2048          ;set history list 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.

When 4DOS/NT 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, see INIQuery.

The *.INI* file has three sections: the first or **global** section, the **[Primary]** section, and the **[Secondary]** section. The global section consists of directives at the beginning of the file, with no section name before them. These directives are effective in all shells. In most cases, this is the only section you will need.

The **[Primary]** and **[Secondary]** sections include directives that are used in primary and secondary shells respectively. Each section is identified by the section name in square brackets on a line by itself. 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.

Sections that begin with any name other than **[Primary]** or **[Secondary]** are ignored.

The SETDOS command can override several of the *.INI* file directives. For example, the cursor shape used by 4DOS/NT 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 under each directive below, and under each option of the SETDOS command.

This example configures certain special characters to match 4DOS, and changes other default settings to suit the user's preferences. All of these settings would also work in 4DOS or 4OS2. Note that the comment for the ColorDir directive is on a separate lines before the directive itself, as no comments are allowed in string directives:

```
PauseOnError = No               ;don't stop on INI errors
CommandSep = ^                  ;4DOS command separator
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
;set directory display colors
colordir=DIRS:bri yel;com exe bat btm cmd:bri whi
```

_DNAME returns the name of the description file (default is DESCRIPT.ION; it can be changed with the DescriptionName .INI directive).

Initialization Directives

The directives in this section control how 4DOS/NT starts and where it looks for its files. The initialization directives are:

4StartPath	Path for 4START and 4EXIT
DirHistory	Size of directory history list
History	Size of history list
INIQuery	Query for each line in 4NT.INI
LocalAliases	Local vs. global aliases
LocalDirHistory	Local vs. global directory history
LocalHistory	Local vs. global history
PauseOnError	Pause on errors in 4NT.INI
WindowState	Initial state for the 4DOS/NT window
WindowX, WindowY, WindowWidth, WindowHeight	Initial size and position of the 4DOS/NT 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 128 to 2048 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.

History = nnnn (1024): Sets the amount of memory allocated to the command history list in bytes. The allowable range of values is 256 to 8192 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 4DOS/NT 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.

LocalAliases = Yes | NO: **No** forces all copies of 4DOS/NT 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 4DOS/NT 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.

WindowState = STANDARD | Maximize | Minimize: Sets the initial state of the 4DOS/NT 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. If you use Maximize or Minimize, you may see the 4DOS/NT 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 4DOS/NT window. The measurements are in pixels or pels. **WindowX and WindowY** refer to the position of the bottom left corner of the window relative to the bottom left corner of the screen.

Configuration Directives

These directives control the way that 4DOS/NT operate. Some can be changed with the SETDOS command while 4DOS/NT is running. Any corresponding SETDOS command is listed in the description of each directive. The configuration directives are:

AmPm	Time display format
BatchEcho	Default batch file echo state
BeepFreq	Default beep frequency
BeepLength	Default beep length
CommandSep	Multiple command separator character
CursorIns	Cursor shape in insert mode
CursorOver	Cursor shape in overstrike mode
DescriptionMax	Maximum length of file descriptions
Descriptions	Enable / disable description processing
EditMode	Editing mode (insert / overstrike)
EscapeChar	4DOS/NT escape character
EvalMax	Max digits after decimal point in @EVAL
EvalMin	Min digits after decimal point in @EVAL
HistCopy	History copy mode
HistLogName	History log file name
HistMin	Minimum command length to save
HistWinColors	History window colors
HistWinHeight	History window height
HistWinLeft	History window left side position
HistWinTop	History window top position
HistWinWidth	History window width
LogName	Log file name
NoClobber	Overwrite protection for output redirection
ParameterChar	Alias / batch file parameter character
Printer	LIST print device
ScreenRows	Screen height
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.

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).

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). Also see SETDOS /C, the %+ internal variable, and 4DOS, 4OS2, and 4DOS/NT 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 overtype 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%. Also see CursorIns and SETDOS /S.

DescriptionMax = nnnn (40): Controls the description length limit for DESCRIBE. The allowable range is 20 to 200 characters.

Descriptions = YES | No: Turns description handling on or off during the file processing commands COPY, DEL, MOVE, and REN. If set to **No**, 4DOS/NT 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. Also see SETDOS /E, the %= internal variable, and 4DOS, 4OS2, and 4DOS/NT 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.

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**, no copy of the command is made. The original copy of the command is always retained at its original position in the list, regardless of the setting of HistCopy.

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.

HistWinColors = Color: Sets the default colors for the command- line and directory history windows. If this directive is not used the colors will be reversed from the current colors on the screen.

HistWinHeight = nn (12): Sets the height of the command-line and directory history windows in lines, including the border. Legal values range from 5 to the height of your screen. Any value which would cause the bottom of the window to be off the screen will be adjusted so that the entire window remains on the screen.

HistWinLeft = nn (40): Sets the horizontal position of the left side of the command-line and directory history windows. Legal values range from 0 (the left edge of the screen) to the number of columns on your screen minus 10. Any value which would cause the right side of a minimum-width window to be off the screen will be adjusted so that the entire window remains on the screen.

HistWinTop = nn (1): Sets the vertical position of the top of the command-line and directory history windows. Legal values range from 0 (the top of the screen) to the number of rows on your screen minus 5. Any value which would cause the bottom of a minimum-height window to be off the screen will be adjusted so that the entire window remains on the screen.

HistWinWidth = nn (36): Sets the width of the command-line and directory history windows in characters, including the border. Legal values range from 10 to the width of your screen. Any value which would cause the right side of the window to be off the screen will be adjusted so that the entire window remains on the screen.

LogName = File: Sets the log file name and path. 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 [**$**]. Also see SETDOS /P. See 4DOS, 4OS2, and 4DOS/NT Compatibility for information on using compatible parameter characters for two or more products..

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.

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 4DOS/NT lower-case style. This directive does not affect the display of filenames on NTFS and HPFS drives, or on FAT drives under Windows NT 3.5 and above. Also see SETDOS /U.

Color Directives

These directives control the colors that 4DOS/NT use for its displays. For complete details on color names see Colors and Color Names. The color directives are:

ColorDir	Directory colors
InputColors	Input colors
ListColors	LIST display colors
ListStatBarColors	LIST status bar colors
SelectColors	SELECT display colors
SelectStatBarColors	SELECT status bar colors
StdColors	Standard display colors

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.

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).

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 4DOS/NT 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

History and @SELECT 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 4DOS/NT 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; 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 4DOS/NT 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; 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 4DOS/NT prompt. The command-line editing keys are:

AddFile	Keeps filename completion entry and adds another
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
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.

CommandEscape = Key (Alt-255): Allows direct entry of a keystroke that would normally be interpreted as an editor command.

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.

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.

History and @SELECT Window Keys

These directives apply only to the command history window, the directory history window, and %@SELECT windows. The History and @SELECT window keys are:

DirWinOpen	Opens the directory history window
HistWinBegin	Moves to the first line of the history window
HistWinDel	Deletes a line from within the history window
HistWinEdit	Moves a line from the history window to the prompt
HistWinEnd	Moves to the last line of the history window
HistWinExec	Executes the selected line in the history window
HistWinOpen	Opens the command history window
NormalHWinKey	Deassigns a history window key

DirWinOpen = Key (Ctrl-PgUp): Opens the directory history window while at the command line.

HistWinBegin = Key (Ctrl-PgUp): Moves to the first line of the history when in the history window.

HistWinDel = Key (Ctrl-D): Deletes a line from within the history window.

HistWinEdit = Key (Ctrl-Enter): Moves a line from the history window to the prompt for editing.

HistWinEnd = Key (Ctrl-PgDn): Moves to the last line of the history when in the history window.

HistWinExec = Key (Enter): Executes the selected line in the history window.

HistWinOpen = Key (PgUp): Brings up the history window while at the command line.

NormalHWinKey = Key: Deassigns a history window key in order to disable the usual meaning of the key within the history window. For additional details see NormalKey.

LIST Keys

These directives are effective only inside the LIST command. The LIST keys are:

ListFind	Prompts and searches 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
ListPrint	Prints the file on LPT1
ListWrap	Toggles LIST's wrap option
NormalListKey	Deassigns a LIST key

ListFind = Key (F): Prompts and searches 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.

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

DescriptionName Set name of descriptions file.

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.).

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.

4DOS/NT Commands

The best way to learn the 4DOS/NT 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	FREE
HISTORY	KEYS	KEYBD	LOG
MEMORY	PROMPT	REBOOT	SETDOS
TIME	VER	VERIFY	VOL

File and directory management:

ATTRIB	COPY	DEL	DESCRIBE
LIST	MOVE	REN	SELECT
TYPE			

Subdirectory management:

CD	CDD	DIR	DIRS
MD	POPD	PUSHD	RD

Input and output:

DRAWBOX	DRAWHLINE	DRAWVLINE	ECHO
ECHOS	INKEY	INPUT	SCREEN
SCRPUT	TEXT	VSCRPUT	

Commands primarily for use in or with batch files and aliases (some work only in batch files; see the individual commands for details):

ALIAS	BEEP	CALL	CANCEL
DELAY	DO	ENDLOCAL	FOR
GLOBAL	GOSUB	GOTO	IF
IFF	LOADBTM	MSGBOX	ON
PAUSE	QUIT	REM	RETURN
SETLOCAL	SHIFT	UNALIAS	

Environment and path commands:

DPATH	ESET	PATH	SET
UNSET			

Other commands:

?	ACTIVATE	DETACH	EXCEPT
EXIT	FFIND	HELP	START
TEE	TIMER	TITLE	WINDOW
Y			

?

Purpose: Display a list of internal commands or prompt for a command.

Format: **? ["prompt text" command]**

Usage

? by itself displays a list of internal commands. If you have disabled a command with SETDOS /I, it will not appear in the list.

If you add prompt text and a command, **?** will display the prompt followed by "(Y/N)?" and wait for the users response. If the user presses "Y" or "y", the command will be executed. If the user presses "N" or "n", the command will be ignored.

For example, the following command might be used in a batch file:

```
? Load the network call netstart.btm
```

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 renames and maximizes the window called "4DOS/NT":

```
[c:\] activate "4DOS/NT" max "4DOS/NT is Great!"
```

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.

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, if you would rather type D instead of DIR /W, you would 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 much faster than it would if 4DOS/NT 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 4DOS/NT or 4DOS/NT. In the text and examples below, we use the 4DOS/NT characters. If you want to use the same aliases under different command processors, see 4DOS, 4OS2, and 4DOS/NT 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 4DOS/NT 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 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:

```
[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 4DOS/NT 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.

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.

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 directive 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 [**@@**]. 4DOS/NT 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$** has a special meaning. 4DOS/NT interprets it to mean "the entire command line, from argument **n** to the end." If **n** is not specified, it has a default value of 1, so **%$** 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 /xmodem /2400
```

the first parameter, **%1**, has the value *c:\comm*. **%2** is *mycomm*, **3**

is */xmodem*, and **%4** is */2400*. The command line expands into these three separate commands:

```
pushd c:\comm
ycomm /xmodem /2400
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
```

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 4DOS/NT. They will not be visible in other shells or other sessions.

With a global alias list, all copies of 4DOS/NT 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 alias 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 4DOS/NT you can share the aliases among all copies of 4DOS/NT running in any session. When you close all 4DOS/NT sessions, the memory for the global alias list is released, and a new, empty alias list is created the next time you start 4DOS/NT.

If you want the alias list to be retained in memory even when no command processor session is running, you need to load the SHRALIAS program, which performs this service for both the global alias list and the global history list. SHRALIAS is supplied with your copy of 4DOS/NT.

To load SHRALIAS, simply run the *SHRALIAS.EXE* program, which is normally installed in the same directory as 4DOS/NT. You may find it convenient to load SHRALIAS from your 4START file.

SHRALIAS runs as a "detached" process, which means it does not have a screen display or accept keyboard input. It is shut down automatically when Windows NT shuts down. To unload SHRALIAS manually, run *SHRALIAS.EXE* with the parameter **/U**.

The UNKNOWN_CMD Alias

If you create an alias with the name **UNKNOWN_CMD**, it will be executed any time 4DOS/NT 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.

Use caution when you create the **UNKNOWN_CMD** alias. If it contains an unknown command, it will be called repeatedly and 4DOS/NT will lock up in an infinite loop.

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.

ATTRIB

Purpose: Change or view file and subdirectory attributes.

Format: **ATTRIB [/A:[[-]rhsda] /D /P /Q /S] [+|-[AHRS]] *files* ...**

files: A file, directory, or list of files or directories on which to operate.

/A:(ttribute select)
/D(irectories)
/P(ause)
/**Q**(uiet)
/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.

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 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), or with the **DIR /T** command.

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
```

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.

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 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.

/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
```

In addition, the **/D** option will keep ATTRIB from appending "*.*" to the end of a directory name and modifying the attributes of all the files in the subdirectory.

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.

/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 insert 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.

4DOS/NT supports batch file nesting up to ten levels deep.

The current ECHO state is inherited by a called batch file.

A called batch file will return to the calling file after processing the last line in the called file, or when a QUIT command is executed. A called batch file should always return in this way, or terminate all batch files with CANCEL. Restarting (or CALLing) the original batch file from within a called file will prevent 4DOS/NT from detecting that you've left the second file, and it 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(**&&** and **||**).

CANCEL

Purpose: Terminate batch file processing.

Format: **CANCEL [*value*]**

value: The exit code from 0 to 255 to return to 4DOS/NT.

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 file(s) 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

Purpose: Display or change the current directory.

Format: **CD [*path* | -]**

or

CHDIR [*path* | -]

path: The directory to change to, including an optional drive name.

See also: CDD, MD, PUSHD, RD, and CDPATH.

Usage

CD and CHDIR are synonyms. You can use either one.

CD lets you navigate through the disk 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.

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.

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 for display in the directory history window.

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.

If CD can't change directly to the specified directory, it will look for the CDPATH variable; see CDPATH for details.

CDD

Purpose: Change the current disk drive and directory.

Format: **CDD *path***

path: The name of the directory (or drive and directory) to change to.

See also: CD, MD, PUSHD, RD, and CDPATH.

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.

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 recorded for display in the directory history window.

If CDD can't change directly to the specified directory, it will look for the CDPATH variable; see CDPATH for details.

CLS

Purpose: Clear the video display and move the cursor to the upper left corner; optionally change the default display colors.

Format: **CLS [[BRIght] *fg* ON [BRIght] *bg***

fg: The new foreground color
bg: The new background color

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
[c:\] cls bri yel on mag bor blu
```

CLS is often used in batch files to clear the screen before displaying text.

See Colors and Color Names for details about colors.

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
```

COPY

Purpose: Copy data between disks, directories, files, or physical hardware devices (such as your printer or serial port).

Format: **COPY [/A:[[-]rhsda] /C /H /M /N /P /Q /R /S /T /U /V] *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:(ttribute select)	**/Q**(uiet)
/B(inary)	**/R**(eplace)
/C(hanged)	**/S**(ubdirectories)
/H(idden)	**/T**(otals)
/M(odified)	**/U**(pdate)
/N(othing)	**/V**(erify)

See also: ATTRIB, MOVE, and REN.

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.

Usage

The COPY command accepts all traditional syntax and options and adds several 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
```

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
```

The way COPY interprets your command line depends on how many arguments (file, directory, or device names) are on the line, and whether the arguments are separated with [**+**] signs or spaces.

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:\] copy a:*.dat
```

If there are two or more arguments on the line and [**+**] signs are not used, 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**.

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. In this case 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, 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
```

To append *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.

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:\
```

COPY maintains the hidden and system attributes of files, but not the read-only attribute. The *destination* file will always have the archive attribute set.

If you COPY files with long filenames from an NTFS volume to a FAT volume, 4DOS/NT will store the *destination* files with their short, FAT-compatible names when running under Windows NT 3.1 (long names are used under Windows NT 3.5 and above). You can view the short names before executing the COPY with the DIR /X command.

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**, **/H**, **/M**, **/N**, **/P**, **/Q**, **/R**, **/S**, **/T**, **/U**, **/V**) 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 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. If you use **/A** with a *destination* filename, a Ctrl-Z will be added to the end of the file (some application programs use the Ctrl-Z to mark the end of a file). **/A** is the default when appending files, or when the *destination* is a device like NUL or PRN, rather than a disk file.

/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 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. You **must** include the colon with this option to distinguish it from the **/A**(SCII) switch, above.

/**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.

/**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.

/**H** (Hidden) Copy all matching files including those with the hidden and/or system attribute set.

/**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 will **not** be cleared after copying; to clear it, 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.

/**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. If you attempt to use COPY /S to copy a subdirectory tree into part of itself, COPY will 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.

DATE

Purpose: Display and optionally change the system date.

Format: **DATE [*mm -dd -yy*]**

mm: The month (1 - 12).
dd: The day (1 - 31).
yy: The year (80 - 99 = 1980 - 1999, or a 4- digit year).

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
Wed  Dec 21, 1994  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 3-16-95
```

You can use hyphens, slashes, or periods to separate the month, day, and year entries. A full 4-digit year can be entered if you wish.

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.

DEL

Purpose: Erase one file, a group of files, or entire subdirectories.

Format: **DEL [/A:[[-]rhsda] /N /P /Q /S /T /X /Y /Z] *file*...**

or

ERASE [/A:[[-]rhsda] /N /P /Q /S /T /X /Y /Z] *file*...

file: The file, subdirectory, or list of files or subdirectories to erase.

/A(ttribute select)	**/S**(ubdirectories)
/F(orce delete)	**/T**(otal)
/N(othing)	**/X** (remove empty subdirectories)
/P(rompt)	**/Y**(es to all prompts)
/Q(uiet)	**/Z**(ap hidden and read-only files)

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
```

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).

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.

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 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.

/**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 is like a GLOBAL DEL, and 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. Unlike **/Q**, the **/T** option will not speed up deletions under DOS.

/**X** (Remove empty subdirectories) Remove empty subdirectories after deleting (only useful when used with **/S**).

/**Y** (Yes) The reverse of **/P** -- it assumes a **Y** response to everything, including deleting an entire subdirectory tree. 4DOS/NT 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.

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
```

A simple loop could make a tone with the BEEP command to get the operator's attention and then DELAY for a few seconds while waiting for the user to respond.

For delays shorter than one second, use the BEEP command with an inaudible frequency (below 20 Hz).

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* ["*description*"] ...**

file: The file or files to operate on.
"*description*": The description to attach to the file.

/A(ttribute select)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

DESCRIBE adds descriptions to files and subdirectories. The descriptions are 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.

Each description can be up to 40 characters long. You can change this limit with the DescriptionMax directive in *4NT.INI*. 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 4DOS/NT if you remove the hidden attribute.) You can change the description file name with the DescriptionName directive in the *4NT.INI* file, and retrieve it with the _DNAME internal variable.

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).

On NTFS and HPFS drives, you will not see file descriptions in a normal DIR display, because DIR must leave space for the long filenames used on these drives. To view the descriptions,

use **DIR /Z** to display the directory in FAT format. See the DIR command for more 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 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.

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, 4DOS/NT will detach a copy of itself to execute the command.

For example, the following command will detach a copy of 4DOS/NT to run the batch file *XYZ.BTM*:

```
[c:\] detach xyz.btm
```

Once the program has started, 4DOS/NT returns to the prompt immediately. It does not wait for a detached program to finish.

DIR

Purpose: Display information about files and subdirectories.

Format: **DIR [/1 /2 /4 /A[[:][-]rhsda] /B /D /E /F /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)
/2 (two columns)
/4 (four columns)
/A(ttribute select)
/B(are)
/D(isable color coding)
/E (upper case)
/F(ull path)
/H(ide dots)
/I (match descriptions)
/J(ustify names)
/K (suppress header)
/L(ower case)
/M (suppress footer)
/N(ew format)
/O(rder)
/P(ause)
/R (disable wRap)
/S(ubdirectories)
/T (aTtribute) or (Time)
/U (sUmmary information)
/V(ertical sort)
/W(ide)
/X (display short names)
/Z (use FAT format)

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, or 5 columns, 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.

You may want to mix several options. 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
```

This example displays all the files on all directories of drive *C*, including hidden and system

files, pausing after each page:

[c:\] dir /s/a/p c:\

DIR allows wildcard characters (* and ?) in the filename. If you don't specify a filename, DIR defaults to *.* (display all non- hidden files and subdirectories in the current directory). To display all of the *.WKS* files in the current directory:

[c:\] dir *.wks

With the **/I** option, DIR can select files to display based on their 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 wild cards like this:

[c:\] dir /i"*test file*"

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

DIR's default output format is different under Windows NT 3.5 and above, where long filenames can be used on on FAT drives. In this case DIR defaults to HPFS / NTFS-style output, normally selected with **/N**; you can switch back to standard FAT-style output with **/Z**. DIR also retains the case in which filenames are stored in Windows NT 3.5 and above, unless you use **/E** or **/L**.

HPFS / NTFS format displays file names on the right (to leave space for long names). In this format file descriptions are not displayed. To display file descriptions on HPFS and NTFS drives, use the **/Z** switch to force the display into the standard FAT format.

You can override the new default output format with an alias if you wish. For example, to use FAT-style output on all drives, define an alias like this:

alias dir=*dir /z

To use FAT-style output on FAT drives only you may want to try the following alias or a variation on it. The alias extracts the drive letter from the first argument to DIR (or uses the current drive if there is no argument), then changes the display format based on whether that argument specifies a FAT or non-FAT drive (enter this on one line):

```
alias dir=`set _drv=%_disk: & if %# != 0
set _drv=%@fstype[%@substr[%@full[%1],0,2]] &
iff "%_drv" == "FAT" then & *dir /z %$ & else & *dir %$ & endiff &
unset /q _drv`
```

You can display the file and subdirectory names in color by setting the COLORDIR environment variable or using the ColorDir directive in your *.INI* file. See Color-Coded Directories for details.

If you are using color-coded directories and attempt to redirect the output of DIR to a character device, such as a serial port or the printer, non-color-coded file names will be displayed on the device but color-coded names may still be displayed on the screen. This will not occur if the output of DIR is redirected to a disk file. To prevent this problem, use the **/D** switch to disable color coding when redirecting the output of DIR to a character device.

When displaying file descriptions, DIR will wrap 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** switch, 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.

If you attempt to redirect the output of DIR to a character device, such as a serial port or the printer, long descriptions will be wrapped at the screen width in the redirected output. If this is not what you want, use **/R** to disable wrapping.

When sorting file names and extensions, 4DOS/NT 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" is after "0" in alphanumeric order). You can defeat this behavior and force a strict alphabetic sort with the **/O:a** option.

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.

DIR can handle directories of any size, limited only by available memory. Memory requirements for DIR are generally not a concern under 4DOS/NT, because of the virtual memory available under these operating systems.

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.

Options

/**1** Single column display -- display the filename, size, date, time, and description. 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.

/**2** Two column display -- display the filename, size, date, and time. If you use **/2** (or **/4**) on an NTFS or HPFS drive, DIR will only display the file names. Also, the number of columns may be reduced to one for names too long to fit on half the screen. Due to these restrictions, **/2** is normally most useful on HPFS drives

when used with **/Z** to force the display to FAT format.

/**4** Four column display -- display the filename and size, in K (kilobytes) or M (megabytes). The note under **/2** above regarding NTFS and HPFS drives applies to **/4** as well.

/**A** (Attribute select) Display only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will display files that do **not** have that attribute set. 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 instead of simply its name and extension.

/**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.

/**H** (Hide dots) Suppress the display of the "." and ".." directories.

/**I** Display filenames by matching text in their descriptions. The text can include wild cards and extended wildcards. The search text must be enclosed in quotation marks. **/I** 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 Use the NTFS/HPFS display format, even if the files are stored on a FAT file system volume. (This is the default for all drives under Windows NT 3.5 and above). 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
- **d** Sort by date and time (oldest first); for NTFS and HPFS drives also see **/T**
- **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) and attributes (if **/A** is used) that you specify on the command line.

/**T** (aTtribute display) Display the filenames and attributes. **/T** is ignored if **/C** or **/O:c** is also used. The attributes are displayed in the format RHSA, with the following meanings:

- **R** Read-only
- **H** Hidden
- **S** System
- **A** Archive

If you wish to add another option after **/T**, you must start the next option with a forward slash. If you dont, 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 an NTFS or HPFS drive should be displayed and used for sorting:

- **a** Last access time
- **c** Creation 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.

/**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 (5 columns on an 80-character wide display).

/**X** Display both the short (8-character name plus 3-character extension) and the long name of files on an HTFS drive.

/**Z** Display an NTFS or HPFS directory (or any directory under Windows NT 3.5) in FAT format. 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.

DIRS

Purpose: Display the current directory stack.

Format: **DIRS**

See also: PUSHD and POPD.

Usage

The PUSHD command adds the current default drive and directory to the directory stack, a list that 4DOS/NT 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 255 characters, enough for 10 to 20 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*

...

[ITERATE]

[LEAVE]

...

ENDDO

n, start, end: An integer between 0 and 2,147,483,647 inclusive, or an internal variable or variable function that evaluates to such a value.
varname: The environment variable that will hold the loop counter.
condition: A test to determine if the loop should be executed.

Usage

DO can only be used in batch files.

DO can be used to create 3 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. You can also specify "forever" for *n* if you wish to create an endless loop. For example:

```
do 5
        beep
enddo
```

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; UNTIL tests it at the end. If you use WHILE, the loop may never be executed (if the condition is false at the start of the loop); if you use UNTIL, the loop will always be executed at least once.

Two special commands, ITERATE and LEAVE, can only 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" != "%val1"
        ...
        if "%var" == "%val2" leave
enddo
```

You can nest DO loops up to 15 levels deep.

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 inside any DO loop.

DPATH

Purpose: Specify the subdirectories which applications will search to find files that are not in the current directory.

Format: **DPATH [*directory* [;*directory...*]]**

directory: The full name of a directory to include in the DPATH (data path) setting.

See also: PATH, SET, and ESET.

Usage

When most Windows NT applications try to open a data file, they look for the file in the current directory first. If they fail to find the file there, they search each of the directories in the DPATH setting in the order that they are included. Internal commands like TYPE do not search the DPATH directories for files.

For example, the following DPATH command directs applications to look for files in this order: the current directory, the *INIT* directory on C, and the *CONFIG* directory on D:

```
[c:\] dpath c:\init;d:\config
```

The listing of directories to be searched can be set or viewed with DPATH. The list is stored as an environment string with the variable name DPATH, and can also be set or viewed with the SET command and edited with the ESET command.

Directory names in the DPATH must be separated with semicolons [**;**]. 4DOS/NT will **not** shift directory names in the DPATH to upper case as it does with those in the PATH setting. If you want the names in the DPATH to be in upper case you must enter them that way.

If you enter DPATH with no parameters, 4DOS/NT displays the current DPATH search list.

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 entire 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 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, 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, 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, the line may not appear on your screen as you expect.

ECHO

Purpose: Display a message, enable or disable batch file or command-line echoing, or display the echo status.

Format: **ECHO [ON | OFF | *message*]**

message: Text to display.

See also: ECHOS, SCREEN, SCRPUT, SETDOS and TEXT.

Usage

4DOS/NT 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 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.

ECHOS

Purpose: Display a message without a trailing carriage return and line feed.

Format: **ECHOS 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).

ENDLOCAL

Purpose: Restore the saved disk drive, directory, environment, and alias list.

Format: **ENDLOCAL**

See also: SETLOCAL.

Usage

The SETLOCAL command in a batch file saves the current disk drive, default directory, all environment variables, and the alias list. ENDLOCAL restores everything that was saved by the previous SETLOCAL command.

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, and their contents 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 4DOS/NT which are using the same alias list.

Option

/**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.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, or size ranges **must** appear immediately after the EXCEPT keyword.

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
```

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.

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 4DOS/NT file processing commands.

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 (see 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 4DOS/NT.

Format: **EXIT [*value*]**

value: The exit code to return (0 - 255).

Usage

EXIT terminates the current copy of 4DOS/NT. 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 4DOS/NT, type:

```
[c:\] exit
```

If you specify a value, EXIT will return that value to the program that started 4DOS/NT. 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. 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 4DOS/NT to perform certain tasks.

FFIND

Purpose: Search for files by name or contents.

Format: **FFIND [/A:[[-]rhsda] /B /C /D[*list*] /E /K /L /M /O[[:][-]acdeginrsu] /P /S /[T|X]"xx" /V] 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)	**/S**(ubdirectories)
/E (upper case display)	**/T"xx"** (text search string)
/K (no headers)	**/V**(erbose)
/L(ine numbers)	**/X["xx"]** (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 version of FFIND 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.

You can use extended wildcards in the search string to increase the flexibility of FFINDs 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**

At times, you may need to search for data that cannot be represented by ASCII characters. You can use FFINDs **/X** option to represent the search string in hexadecimal format. With **/X**, the search must be represented by pairs of hexadecimal digits separated by spaces; a search of this type is always case-sensitive (41 63 65 is the hex code for "Ace"):

c:\> **ffind /x"41 63 65" *.txt**

You can use FFINDs other options to further specify the files for which you are searching and to modify the way in which the output is displayed.

Options

/A: (Attribute select) -- Find only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will display files that do **not** have that attribute set. 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-insenitive 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 the default 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 ...**

/E Display filenames in the traditional upper case; also see SETDOS /U and the UpperCase directive in *4NT.INI*.

/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 FFINDs 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 NTFS and HPFS drives also see **/T**
- **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.

/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. FFINDs 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"] (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 (also in 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 hexadecimal offsets 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 hexadecimal offset is displayed

instead of the text:

```
c:\>ffind /t"hello" /x *.btm
---- c:\test.btm
Offset: 1A

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] /H] *%var* IN ([@]*set*) DO] *command* ...**

%var: The variable to be used in the command ("FOR variable").

set: A set of values for the variable.
command: A command or group of commands to be executed for each value of the variable.

/A(ttribute select) **/H**(ide dots)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, or size ranges **must** appear immediately after the FOR keyword.

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.

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.)

The set can include multiple files or an include list, like this:

```
for %x in (d:\*.txt;*.doc;*.asc) do type %x
```

If the set includes filenames, the file list can be further refined by using date, time, and size ranges. The range must be placed immediately after the word FOR. The range 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, 1994:

```
for /[d10-4-94,+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.

The set can also be made up of text instead of file names. For example, to display the free space on drives *C:*, *D:*, and *E:*, you could use:

```
for %drive in (c d e) do free %drive:
```

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 [**/**]).

You can also set the FOR variable equal to each line 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 (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.

You can use FOR to process the output of a command by using a pipe. To do so, use **@CON** as the file name. For example, this command creates a list of the names of all *.MSG* files in date/time order, then calls the MSGPROC batch file for each file:

```
dir /b /od *.msg | for %fn in (@con) do call msgproc %fn
```

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 use *%%* if it 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
```

You can use command grouping to execute multiple commands for each element in the list. 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:

[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. However, for compatibility with *CMD.EXE*, single-character FOR variables do not overwrite existing environment variables with the same name. As a result, 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.

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 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!

/H (Hide dots) -- Suppress the assignment of the "." and ".." directories to the *var*.

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:
```

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 command to continue in additional subdirectories even if it returns an error in a previous subdirectory. Even if you use **/I**, GLOBAL will halt execution in response to **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.

4DOS/NT 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. 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
```

If the *label* doesn't exist, the batch file is terminated with the error message "Label not found."

GOSUB saves the IFF state, so IFF 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.

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:\
```

If the *label* doesn't 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.

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 4DOS/NT. The 4DOS/NT 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***

/A(dd) **/P(**ause)
/F(ree) **/R**(ead)

See also: LOG.

Usage

4DOS/NT 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 8192 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 4DOS/NT) or globally (all copies of 4DOS/NT 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 is 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 with the HistMin directive in the *.INI* file.

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).

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 or other startup 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 save the history before shutting off the system.

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 [NOT] *condition* [.AND. | .OR. | .XOR. [NOT] *condition* ...] *command***

condition: A test to determine if the command should be executed.
command: The command to execute if the condition is true.

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 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 alias names 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.

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.)

Conditions

The following conditional tests 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.

Spaces are required on either side of the test condition in all cases, except == which will work with or without spaces around it.

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). 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*

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.

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. 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.
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.
ISWINDOW "title"	If the window with the title exists, the condition is true. Double quotes must be used around the title.

Combining Tests

You can negate the result of any test with **NOT**, and combine tests of any type with **.AND.**, **.OR.**, and **.XOR.** Test conditions are always scanned from left to right -- there is no implied order of precedence, as there is in some programming languages.

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.

Using the IF Tests

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 point), a numeric comparison is used. If either value does not begin with a numeric character, 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, it forces IF to do 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 to the left of the decimal point may not exceed 2,147,483,648 (the maximum possible 32-bit positive integer). The number of digits to the right of the decimal point is limited only by the length of the command line.

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 "%@instr[0,2,%@line[myfile,0]]"=="GO" call go.btm
```

This batch file fragment tests for the existence of *A:\JAN.DOC* before copying it to drive C.

```
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
```

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.

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 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. In both cases, after the selected *commands* 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 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*.

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.

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 the Key Code Tables for a list of extended key codes.

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. In a batch file you can handle **Ctrl-C** and **Ctrl-Break** yourself with the ON BREAK 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 INKEY 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, 4DOS/NT 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. 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 users 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. In a batch file you can handle **Ctrl-C** and **Ctrl-Break** yourself with the ON BREAK 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. 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 users 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 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 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, 4DOS/NT 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 /S /W /X] *file*...**

file: A file or list of files to display.

/A(ttribute select)
/H(igh bit off)
/S(tandard input)
/W(rap)
/X (heX display mode)

See also: TYPE.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

Usage

LIST provides a much faster and more flexible way to view a file than TYPE, without the overhead of loading and using a text editor.

LIST is most often used for displaying ASCII text files. Most other files contain non-alphabetic characters and may be unreadable, except in hex mode.

For example, to display a file called *MEMO.DOC*:

```
[c:\] list memo.doc
```

LIST uses the cursor pad to scroll through the file. The following keys have special meanings:

Space	Display the next page of the file (same as **PgDn**).
Home	Display the first page of the file.
End	Display the last page of the file.
Esc	Exit the current file.
Ctrl-C	Quit LIST.
	Scroll up one line.
-	Scroll down one line.
¬	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.

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.

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** and **N** are not case sensitive. However, if the display is currently in hexadecimal mode and you press **F**, you will be prompted for whether you want to search in hexadecimal as well. 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.

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". See Wildcards for complete information on wildcards.

LIST saves the search string used by **F** and **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.

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.

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 Printer directive in the *.INI* file.

Most of the LIST keystrokes can be reassigned with key mapping directives in the *.INI* file .

You can set the colors used by LIST with the 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.

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 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.

/**S** (Standard input) Read from standard input rather than a file. This allows you to redirect command output and view it with LIST. For example, to use LIST to display the output of DIR:

```
[c:\] dir | list /s
```

/**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

4DOS/NT recognizes two kinds of batch files: *.BAT* or *.CMD*, and *.BTM*. Batch files executing in BTM mode run two to five 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.

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. 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, but it does not record the date and time for each command. In addition, it records commands before aliases and variables are expanded.

By default, LOG writes to the file *4NTLOG* in the root directory of the boot drive. The default file name for LOG /H is *4NTHLOG*.

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"
```

Entering LOG or LOG /H with no parameters now displays the name of the log file in addition to the log status (ON or OFF):

```
c:\> log
LOG (C:\4NTLOG) is OFF
```

The LOG file format looks like this:

```
[mm-dd-yy   hh:mm:ss]   command
```

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 turns on (or off) the history log, which saves commands without the time and date stamp. 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. You can set the default log file names when 4DOS/NT starts with the LogName and HistLogName directives in the *.INI* file.

MD

Purpose: Create a subdirectory.

Format: **MD [/S] *pathname*...**

or

MKDIR [/S] *pathname*...

pathname: The name of one or more directories to create.

/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 pathname 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 [**..**].

Option

/**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
```

MEMORY

Purpose: Display the amount and status of system memory.

Format: **MEMORY**

Usage

MEMORY lists the memory load, 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.

MOVE

Purpose: Move files to a new directory and drive.

Format: **MOVE [/A:[[-]rhsda] /C /D /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(ttribute select)
/C(hanged)
/D(irectory)
/H(idden and system)
/**M**(odified files)
/N(othing)
/P(rompt)
/Q(uiet)
/R(eplace)
/S(ubdirectory tree)
/T(otal)
/U(pdate)
/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.

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.)

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 target is not a directory, MOVE will display an error message and exit):

```
[c:\] move *.wks *.txt c:\finance\myfiles
```

You cannot move a file to a character device like the printer, or to itself.

When you move files to another directory, if you add a backslash [\] to the end of the

destination 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 all *source* files to that name. The **/D** option performs the same function but will also prompt to see if you want to create the *destination* directory if it doesn't exist.

Be careful when you use MOVE with the SELECT command. If you SELECT multiple files and the target 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 target 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 target 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).

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 (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 stores deleted files in a hidden directory, or if, under a multitasking system, 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 *source* file attributes are not changed.

If you MOVE files with long filenames from an NTFS volume to a FAT volume, 4DOS/NT will now store the *destination* files with their short, FAT-compatible names when running under Windows NT 3.1 (long names are used under Windows NT 3.5 and above). You can view the short names before executing the MOVE with the DIR /X command.

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 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.

/**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.

/**H** (Hidden) Move all files, including hidden and system files.

/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 you attempt to use **/S** to move a subdirectory tree into part of itself, MOVE will display an error message and halt.

/**T** (Total) Don't display filenames as they are moved, but display the total number of files deleted 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.

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. 4DOS/NT 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 4DOS/NT variable **%_?**. Be sure to save the return value in another variable or test it immediately; 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.

ON

Purpose: Execute a command in a batch file when a specific condition occurs.

Format: **ON BREAK [*command*]**

or

ON ERROR [*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 *command* is automatically executed.

ON BREAK will execute its *command* if the user presses **Ctrl- C** or **Ctrl-Break**.

ON ERROR will execute its *command* after any command processor or operating system error (including critical errors). That is, ON ERROR will detect errors such as a disk write error, and 4DOS/NT errors such as a COPY command that fails to copy any files, or the use of an unacceptable command option.

ON BREAK and ON ERROR are independent of each other. You can use either one, or both, in any batch file.

Each time ON BREAK or ON ERROR 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 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 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 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, you may find it useful to use internal variables, particularly %_? and %_SYSERR, to help determine the cause of the error.

Caution: If a break or error occurs while the *command* specified in ON BREAK or ON ERROR 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, since such errors can cause an infinite loop.

PATH

Purpose: Display or alter the list of directories that 4DOS/NT 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 4DOS/NT 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, 4DOS/NT 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 can be set or viewed with the PATH command. The list 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.

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.

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 4DOS/NT 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 4DOS/NT 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 and PUSHD.

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 for display 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 1-06-95* (the month, day, and year are formatted according to your current country settings).

D Current date, in the format: *Fri Jan 6, 1995*.

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).

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:* (including drive letter), in lower case. (Uses the actual case of the directory name as stored on the disk for HPFS and NTFS drives, and FAT drives under Windows NT 3.5 and above).

Xd: Current directory on drive *d:* (including drive letter), in upper case.

z Current shell nesting level; the primary command processor is shell 0.

$ 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   Dec 2, 1994 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   Dec 2, 1994 10:29:19
[c:\]
```

The 4DOS/NT prompt can be set in 4START, or in any batch file that runs when 4DOS/NT starts. The 4DOS/NT 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 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 [*pathname*]**

pathname: The name of the new default drive and directory.

See also: DIRS, POPD and the CDPATH environment variable.

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 *pathname*.

If a *pathname* 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 *pathname* 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:\]
```

Directory changes made with PUSHD are recorded for display in the directory history window.

The directory stack can hold up to 255 characters, or about 10 to 20 entries (depending on the length of the names). If you exceed this limit, the oldest entry is removed before adding a new entry.

If PUSHD can't change directly to the specified directory, it will look for the CDPATH variable; see CDPATH for details.

QUIT

Purpose: Terminate the current batch file.

Format: **QUIT [*value*]**

value: The exit code from 0 to 255 to return to 4DOS/NT 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 (see the IF command, and the %? variable) to that value.

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

Purpose: Remove one or more subdirectories.

Format: **RD [/S] *pathname* ...**

or

RMDIR [/S] *pathname* ...

pathname: The name of a subdirectory 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 *pathname* 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 *pathname*.

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 does not prompt for permission before deleting files and subdirectories, and 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 completely restart your computer. It normally performs a warm reboot, which is comparable to pressing Ctrl-Alt-Delete. 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. 4DOS/NT 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 Logoff choice on the Program Manager's File menu in Windows NT.

/**S** (Shutdown) Shut down the system, but do not reboot. This option is equivalent to the Shutdown choice on the Program Manager's File menu in Windows NT.

/**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. 4DOS/NT 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 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 4DOS/NT is to enter **>filename** as a command, with no actual command before the [**>**] redirection character.)

REN

Purpose: Rename files or subdirectories.

Format: **REN [/A:[[-]rhsda] /N /P /Q /S /T] *old_name*... *new_name***

or

RENAME [/A:[[-]rhsda] /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(ttribute select)	**/Q**(uiet)
/N(othing)	**/S**(ubdirectory)
/P(rompt)	**/T**(otal)

See also: COPY and MOVE.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists.

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 simply 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
```

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
```

When *new_name* refers to a file or files (rather than a directory), the file(s) must not already exist. Also, 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 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.

/**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.

/**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**

See also: GOSUB.

Usage

4DOS/NT 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.

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
```

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 menu displays, logos, etc. 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!
```

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: CLS, 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 allows you to specify the display colors. It always leaves the cursor in its current position.

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. 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.

The following batch file fragment displays part of a menu, in color:

```
cls white on blue
scrput 6 20 bri red on blu 1 - Word Processing
scrput 7 20 bri yel on blu 2 - Spreadsheet
```

SELECT

Purpose: Interactively select files for a command.

Format: **SELECT [/A[:][-]rhsda /D /E /H /I"text" /O[:][-]adeginrsu /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)
/D(isable color coding)
/E (use upper case)
/H(ide dots)
/I (match descriptions)
/O(rder)
/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.

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 its command-line arguments.

SELECT uses the cursor up, cursor down, PgUp, and PgDn keys to scroll through the file list. The cursor right and cursor left keys let you scroll through file descriptions that are longer than the screen width. Press the **L** key to view the current highlighted file with LIST. When you exit from LIST, the SELECT screen will be restored. Use the **+** key or the **spacebar** to select a file (or unselect a marked file), and the **-** key to unselect a file. The ***** key will reverse all of the current marks (excluding subdirectories), and the **/** key will unmark everything. After marking the files, press **Enter** to execute the command.

You can select a single file by moving the scroll bar to the filename and pressing **Enter** without marking any other files.

To skip the files listed in the current display and go on to the next file specification inside the parentheses or brackets (if any), press the **Esc** key. To cancel the current SELECT command entirely, press **Ctrl-C** or **Ctrl-Break**.

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* and *.EXE* files on the current drive. It will then invoke the COPY command to copy each file you select to drive A:. You will be able to select first from a list of all *.COM* files in the current directory, and then from a list of all *.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) is no more than 1,023 characters long. 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.

The list of files from which you wish to select can be further refined by using date, time, and size ranges. The range 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.

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.

You can set the default colors used by SELECT 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 display colors.

You can display the filenames in color by setting the COLORDIR environment variable or using the ColorDir directive in your *.INI* file. See Color-Coded Directories for details. To disable directory color coding within SELECT, use the **/D** option.

When displaying descriptions, 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 screen horizontally and view the additional text.

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 wild cards as part of the text.

When sorting file names and extensions for the SELECT display, 4DOS/NT 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) Display only those files that have the specified attribute set. Preceding the attribute character with a minus [**-**] will display files that do **not** have that attribute set. Attributes can also be combined. 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.

/**D** (Disable color coding) Temporarily turn off directory color coding within SELECT.

/**E** (use upper case) Display filenames in the traditional upper case format; 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 wild cards and extended wildcards. The search text must be enclosed in quotation marks. **/I** will be ignored if **/C** or **/O:c** is also used.

/**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
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

/**Z** Display NTFS and HPFS filenames in FAT format. 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 [/P /R *filename*...] [*name* =][*value*]]**

filename: The name of a file containing variable definitions.
name: The name of the environment variable to define or modify.
value: The new value for the variable.

/P(ause) **/R**(ead from file)

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:\4DOS/NT\$OS2.EXE
PATH=C:\;C:\OS2;C:\OS2\SYSTEM;C:\UTIL
CMDLINE=C:\4DOS/NT\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 4DOS/NT. 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 4DOS/NT 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 4DOS/NT the size of the environment is set automatically, and increased as necessary as you add variables.

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.

/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 4DOS/NT: The file is in the same format as the SET display, 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 4DOS/NT configuration.

Format: **SETDOS [/C? /D /E? /Fn.n /I+|- command /M? /N? /P? /R? S?:? /U? /V? /X[+|-]n /Y]**

/C(ompound)	**/P**(arameter character)
/D(escriptions)	**/R**(ows)
/E(scape character)	**/S**(hape of cursor)
/**F**(ormat for @EVAL)	**/U**(pper case)
/I(nternal commands)	**/V**(erbose)
/M(ode for editing)	**/X** (expansion, special characters)
/N(o clobber)	**/Y** (single step)

Usage

SETDOS allows you to customize certain aspects of 4DOS/NT 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 4DOS/NT executes the configuration directives in the *.INI* file. The name of the corresponding directive is listed with each option below; if none is listed, that option cannot be set 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 4DOS/NT 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) The COMPOUND 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. This example changes the COMPOUND character to a tilde [~]:

```
[c:\] setdos /c~
```

If you want to share batch files or aliases between 4DOS and 4DOS/NT or 4DOS/NT, see the %+ variable, which retrieves the current command separator, and 4DOS, 4OS2, and 4DOS/NT Compatibility for details on using compatible command separators for all the products you use. Also see the CommandSep directive.

/**D** (Descriptions) The DESCRIPTIONS 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.

/**E** (Escape character) The ESCAPE 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. Certain characters (**b**, **c**, **e**, **f**, **n**, **r**, **s**, and **t)** have special meanings when immediately preceded by the escape character.

If you want to share batch files or aliases between 4DOS and 4DOS/NT or 4DOS/NT, see the %= variable, which retrieves the current escape character, and 4DOS, 4OS2, and 4DOS/NT Compatibility for details on using compatible escape characters for all the products you use. Also see the EscapeChar directive.

/F (Format for @EVAL) The FORMAT 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. By default, the minimum precision to the right of the decimal point is 0.

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. 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.

/**I** (Internal) The INTERNAL 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 4DOS/NT to use an external command:

```
[c:\] setdos /i-list
```

/**M** (Mode) The MODE 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) The NOCLOBBER 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 [**$**].

If you want to share batch files or aliases between 4DOS and 4DOS/NT or

4DOS/NT, see 4DOS, 4OS2, and 4DOS/NT Compatibility for details on selecting compatible parameter characters for all the products you use. Also see the ParameterChar directive.

/**R** (Rows) The ROWS option sets the number of screen rows used by the video display. Normally 4DOS/NT detects the screen size, but if you have a non-standard display you may need to set it explicitly. This option does not affect screen scrolling (which is controlled by your video driver). It is used only for LIST, SELECT, the paged output options (*i.e.*, TYPE /P), and error checking in the screen output commands. Also see the ScreenRows directive.

/**S** (Shape) The SHAPE 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 (an underscore cursor for overstrike mode, and a 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. Also see the CursorOver and CursorIns directives.

/**U** (Upper) The UPPER 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. The **/U** setting is ignored for NTFS and HPFS filenames, and FAT filenames under Windows NT 3.5 and above. NTFS and HPFS names are always displayed in the case in which they are stored. Also see the UpperCase directive.

/**V** (Verbose) The VERBOSE 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 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 4DOS/NT 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** (Single step) **/Y1** enables single-stepping through a batch file. Each command is displayed on the screen along with a **Y/N/R** (yes / no / remainder) prompt. Press **Y** to execute the command, **N** to omit the command and go on to the next, or **R** or **Esc** to execute the remainder of the batch file (up to the next SETDOS /Y1 command). You may also press **Ctrl-C** or **Ctrl-Break** to terminate the batch file.

Batch file single stepping is disabled each time 4DOS/NT returns to the command prompt. This means you cannot enter the SETDOS /Y1 command at the prompt, press Enter, and start a batch file in single step mode at the next prompt. However you can enable single step operation and run a batch file from the prompt if you enter both commands on one line. For example, this command runs *FILECOMP.CMD* with single step enabled:

```
[c:\] setdos /y1 & filecomp.cmd
```

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

DescriptionName = File: Sets the filename to use instead of *DESCRIPT.ION*. This is intended primarily for BBS sysops who wanted to use FILES.BBS. *Use this directive with caution* as changing the name will make it difficult to transfer file descriptions to other systems!

SETLOCAL

Purpose: Save a copy of the current disk drive, directory, environment, and alias list.

Format: **SETLOCAL**

See also: ENDLOCAL.

Usage

SETLOCAL is used in batch files to save the default disk drive and directory, the environment, and the alias list to a reserved block of memory. You can then change their values and later restore the original values with the ENDLOCAL 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, modifies a variable, runs a program, and then restores the original values:

```
setlocal
unalias *
cdd d:\test
set path=c:\;c:\dos;c:\util
rem run some program here
endlocal
```

SETLOCAL and ENDLOCAL are not nestable within a batch file. However, you can have multiple 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 drive, directory, aliases, and environment variables as they were prior to any unterminated SETLOCAL.

SHIFT

Purpose: Allows the use of more than 127 parameters in a batch file.

Format: **SHIFT [*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. 4DOS/NT 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).

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=row,col,width,height /REALTIME /SEPARATE /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)
/C(lose when done)
/D(irectory)
/HIGH (priority)
/I(nherit environment)
/INV(isible)
/K(eep when done)
/L(ocal lists)
/LA (local aliases)
/**LD** (local directory history)
/LH (local history list)
/MAX(imized)
/MIN(imized)
/N(o *4NT.EXE*)
/NORMAL (priority)
/PGM (program name)
/POS(ition of window)
/REALTIME (priority)
/**SEPARATE** (virtual machine)
/SIZE (of screen buffer)
/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.

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 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.

/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 4DOS/NT with local alias, history, and directory history lists. This option combines the effects of **/LA, /LD,** and **/LH** (below).

/LA (Local Alias list) Start 4DOS/NT with a local alias list. See ALIAS for information on local and global aliases.

/LD (Local Directory history list) Start 4DOS/NT with a local directory history list. See Directory History Window for information on local and global directory history.

/LH (Local History list) Start 4DOS/NT 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. 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=row, col, width, height** where the values are specified in pixels or pels. **Row** and **col** 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 (Windows NT 3.5 and above only).

/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.

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 "4DOS/NT", make a copy of the matching lines in *4.DAT*, sort the lines, and write them to the output file *4O.DAT*:

```
[c:\] find "4DOS/NT" doc | tee 4.dat | sort > 4o.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 4DOS/NT, the programs on the two ends of the pipe run simultaneously, not sequentially as in 4DOS.

See Piping for more information on pipes.

Option

/**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
endtext
```

TIME

Purpose: Display or set the current system time.

Format: **TIME [*hh* [:mm :ss]]] [AM | PM]**

hh: The hour (0 - 23).
mm: The minute (0 - 59).
ss: The second (0 - 59).

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
Wed   Dec 21, 1994   9:30:10
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.

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.

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 4DOS/NT window. It is included only for compatibility with *CMD.EXE*. You can also change the window title with the WINDOW command or the ACTIVATE command.

The title text must be enclosed in double quotes. The quotes will not appear as part of the actual title.

To change the title of the current window to "4DOS for Windows NT":

```
[c:\] title "4DOS for Windows NT"
```

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(ttribute 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.

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.

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 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] *alias*...**

or

UNALIAS *

alias: One or more aliases to remove from memory.

/Q(uiet)

See also: ALIAS and ESET.

Usage

4DOS/NT 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 *
```

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.

UNSET

Purpose: Remove variables from the environment.

Format: **UNSET [/Q] *name*...**

or

UNSET *

name: One or more variables to remove from the environment.

/Q(uiet)

See also: ESET and SET.

Usage

UNSET removes one or more variables from the environment. 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 is often used in conjunction with the SETLOCAL and ENDLOCAL commands in order to clear the environment of variables that may cause problems for some applications.

For more information on environment variables, see the SET command and the general discussion of the environment.

Use caution when removing environment variables, and especially when using UNSET *. Many programs will not work properly without certain environment variables; for example, 4DOS/NT uses PATH and DPATH.

Options

/Q (Quiet) Prevents UNSET from displaying an error message if one or more of the variables 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.

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 period, and a one- or two-digit minor version number. The VER command displays both version numbers:

```
[c:\] ver
4DOS/NT 2.5     Windows NT Version is 3.5
```

Option

/**R** (Revision level) Display the 4DOS/NT and Windows NT internal revision levels, plus your 4DOS/NT 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. 4DOS/NT 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. It always leaves the cursor in its current position.

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. 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.

VSCRPUT checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

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=row,col,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 row and column values of the **/POS** option select the window's origin (from the top 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.

Only one WINDOW option can be used at a time. To make multiple changes in the window state (for example, to maximize the window and change its title) you must issue the WINDOW command once for each change.

Option

/POS Set the window screen position and size. The syntax is **/POS=row, col, width, height**, where the values are specified in pixels or pels. **Row** and **col** 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.

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.

Error Messages

This section lists error messages generated by 4DOS/NT, 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. 4DOS/NT 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.

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 deep. Correct your alias list.

Bad disk unit: Generally caused by a disk drive hardware failure.

Batch file missing: 4DOS/NT 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. 4DOS/NT performs full path and filename expansion before copying to ensure that files aren't inadvertently destroyed.

Can't create: 4DOS/NT 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: 4DOS/NT can't delete the specified file or directory. The disk is probably write protected.

Can't get directory: 4DOS/NT can't read the directory. The disk drive is probably not ready.

Can't make directory entry: 4DOS/NT 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: 4DOS/NT 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.

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 target. 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 floppy disk drive door is open. Close the door 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. A more specific error message follows.

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 4DOS/NT 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 exists: The requested output file already exists, and 4DOS/NT 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: 4DOS/NT couldn't find the specified file. Check the spelling and path name.

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; the problems described there can sometimes cause a general failure rather than a data error.

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 date: An invalid date was entered. Check the syntax and reenter.

Invalid directive name: 4DOS/NT 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: 4DOS/NT 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: 4DOS/NT ran out of room to store keystroke substitutions entered in the *4NT.INI* file. Reduce the number of key substitutions or contact JP Software 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: 4DOS/NT 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: 4DOS/NT 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 room for INI file name: 4DOS/NT 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: 4DOS/NT 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 (LPT1, LPT2, or LPT3). 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.

Sector not found: Disk error, usually caused by a bad or unformatted disk.

Seek error: Windows NT can't seek to the proper location on the disk. This is generally caused by a bad disk or drive.

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.

String area overflow: 4DOS/NT 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 4DOS/NT 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).

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.

Key Code Tables

The tables in this section 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.

To represent the text you type, computers must translate each letter to and from a number. The code used by all PC-compatible computers for this translation is called **ASCII**. Function keys, cursor keys, and Alt keys generate **scan codes** indicating which key was pressed, but not ASCII codes. This section includes a table showing the codes for each key on your keyboard, and an explanation of how key codes work.

For more information, see:

Key Codes and Scan Codes Table

Key Codes and Scan Codes Explanation

Key Codes and Scan Codes Table

(For more details on key codes and scan codes, see the Key Codes and Scan Codes Explanation.)

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, 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

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt 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

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt 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

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt 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

Key Cap Symbol	Scan Code	ASCII Code	Shift Scan Code	Shift ASCII Code	Ctrl Scan Code	Ctrl ASCII Code	Alt Scan 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 4DOS/NT see the *4NT.INI* file key mapping directives, keystroke aliases, and INKEY.

Most 4DOS/NT 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 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 manual.

Support

You can contact JP Software at any of the following addresses. Our normal business hours are 9:00 AM to 5:00 PM weekdays, eastern US time.

By mail:

JP Software Inc.
P.O. Box 1470
East Arlington, MA 02174
USA

By telephone / fax:

Voice	(617) 646-3975
Fax	(617) 646-0904
Order Line	(800) 368-8777 (orders only, USA only)

Electronically:

CompuServe

Customer Service 75020,244
Technical Support, GO JPSOFT or GO PCVENB (section 10), User ID 75300,1215

Internet

Customer Service 75020.244@compuserve.com
Technical Support 75300.1215@compuserve.com

BBS Support

Via Channel 1 BBS, Boston, 617-354-5776 at 2,400 - 14,400 baud, no parity, 8 data bits, 1 stop bit.

Technical support is available via public electronic support conferences, private electronic mail, telephone, fax, and mail.

Often the best way to contact us for support is in one of the following public electronic support conferences. The numbers in parentheses indicate the usual delay, in business days, to receive a reply to a message.

CompuServe / ZiffNet: Primary support is via the JP Software section of the CompuServe PCVENB forum (GO JPSOFT or GO PCVENB, section 10, "JP Software") (1 day).

Bulletin Boards: Primary support is via the Channel 1 BBS, Boston, MA (1 - 3 days; see above for access details). Messages may be left in any of the "4DOS" conferences; check the online list for exact conference numbers. Support is also available from many local BBSes via the "4DOS" conferences on the RIME, ILink, SmartNet, and FidoNet BBS Networks (3-5 days).

Before contacting us for support, please check the manuals 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 4DOS/NT to run properly, either alone or with your particular hardware or software, see the Introduction and Installation Guide, and the APPNOTES.DOC file. Also look

through the README.DOC and UPDATxxx.DOC files, as they may contain updates to the manual or other important information ("xxx" is the version number).

If you do need to contact us for support, it helps if you can give us some basic information:

> 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 went wrong? 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?
>
> What computer and operating system version are you using?
>
> Are you running a network? If so, which one, and which version?
>
> What are the contents of any startup files you use (such as *4START*, *4EXIT*, and *4NT.INI*), 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?